Universidad Politécnica de Madrid
Facultad de Informática

Master of Science in High-End Computing for Sciences and Engineering

Master Thesis

Optimisation of Algorithms to Compute Information Theoretic Indexes

AUTHOR: Ángel Esquinas Fernández
TUTOR: Antonio García Dopico

Contents

List of Figures

Chapter 1

Introduction

Analysis of big amount of data is a field with many years of research. It is centred in getting significant values, to make it easier to understand and interpret data. Being the analysis of interdependence between time series an important field of research, mainly as a result of advances in the characterization of dynamical systems from the signals they produce.

In the medicine sphere, it is easy to find many researches that try to understand the brain behaviour, its operation mode and its internal connections. The human brain comprises approximately 10^{11} neurons, each of which makes about 103 synaptic connections. This huge number of connections between individual processing elements provides the fundamental substrate for neuronal ensembles to become transiently synchronized or functionally connected [4]. A similar complex network configuration and dynamics can also be found at the macroscopic scales of systems neuroscience and brain imaging[3]. The emergence of dynamically coupled cell assemblies represents the neurophysiological substrate for cognitive function such as perception, learning, thinking[23]. Understanding the complex network organization of the brain on the basis of neuroimaging data represents one of the most impervious challenges for systems neuroscience. Brain connectivity is an elusive concept that refers to different interrelated aspects of brain organization: structural, functional connectivity (FC) and effective connectivity (EC). Structural connectivity refers to a network of physical connections linking sets of neurons, it is the anatomical structur of brain networks. However, FC refers to the statistical dependence between the signals stemming from two distinct units within a nervous system, while EC refers to the causal interactions between them. This research opens the door to try to resolve diseases related with the brain, like Parkinson's disease, senile dementia, mild cognitive impairment, etc. One of the most important project associated with Alzheimer's research and other diseases are enclosed in the European project called *Blue Brain*[16]. The center for Biomedical Technology (CTB) of Universidad Pólitecnica de Madrid (UPM) forms part of the project. The CTB researches have developed a magnetoencephalography (MEG) data processing tool that allow to visualise and analyse data in an intuitive way. This tool receives the name of HERMES[19], and it is presented in this document.

MEG is technique for mapping brain activity by recording magnetic fields produced by electrical currents occurring naturally in the brain, using very sensitive magnetometers. Arrays of superconducting quantum interference devices (SQUIDS) are currently the most common magnetometers. It allows to research into perceptual and cognitive brain processes, determining the function of various parts of the brain, etc. But electroencefalography (EEG) that measures bioelectric process and have a similar spatial resolution to MEG, have a limited spatial resolution. Although EEG and MEG signals originate from the same neurophysiological processes, EEG signals are affected by the electrical resistance of the different tissues that the signals need to go through to reach the external electrode. On the contrary, MEG records primary electrical activity, whose magnetic fields does not undergo attenuation, distortion problems or conductivity changes.

The principal goal of this work is to improve HERMES tool. This is achieved optimising two algorithms used by HERMES for time series analysis based in information theory: the first is used to estimate mutual information[6] between two signals; the second is used to estimate transfer entropy[21] between two signals, reducing the required time to get results, because they are currently high. Also, we want that the algorithms are able to execute in different operative systems with HERMES tool.

Mutual Information (MI) is used in a variate set of fields: detection of phase synchronisation in time series analysis, noise clean of images, events analysis in stock market, etc. In HERMES, the MI is used to get the correlation activity of different brain parts. In a MEG, it is easy to find many sensors placed everywhere in cranial convexity, each sensor collects one signal at a time. This signals are denominated "Channel", and different numbers of samples could be recorded depending on the sampling frequency and duration of the session. Transfer entropy (TE) is a non-parametric statistic measuring the amount of directed transfer information between two random processes. TE has been used for estimation of functional connectivity of neurons[24] and social influence in social networks.

The algorithms developed within the framework of this work are built over Tim[1] and Pastel[7] libraries. A software layer between TIM and HERMES has been created as a result of this work, This library receives the name of HermesTim. With this library, we have reduced the time necessary to estimate the mutual information or transfer entropy of a set of channels. HermesTim library allows a better scalability in multiprocessor systems. We use OpenMP[20, 5] to parallelise sections of code to be able to get this improvement in performance.

HermesTim is the software library implemented as a result of this work. HermesTim is a multi-platform library written in C++ that provides a MATLAB interface. The MATLAB interface is a requirement of HERMES to be able to use other libraries.

1.1 Hermes

Measure synchronisation tools, (from spanish, HERramientas de MEdidas de Sincronización) (HERMES) is a toolbox for the MATLAB environment, which is designed to study functional and effective brain connectivity from neurophysiological data such as multivariate electroencephalography (EEG) and/or MEG records. HERMES encompasses several of the most common indexes for the assesment of FC and EC. It includes visualization tools and statistical methods to address the problem of multiple comparisons, which are very useful tools for the analysis of connectivity in multivariate neuroimage datasets.

HERMES has to be launched from the MATLAB environment. The simplest and most straightforward way of using it is through its graphic user interface (GUI) (figure 1.1.

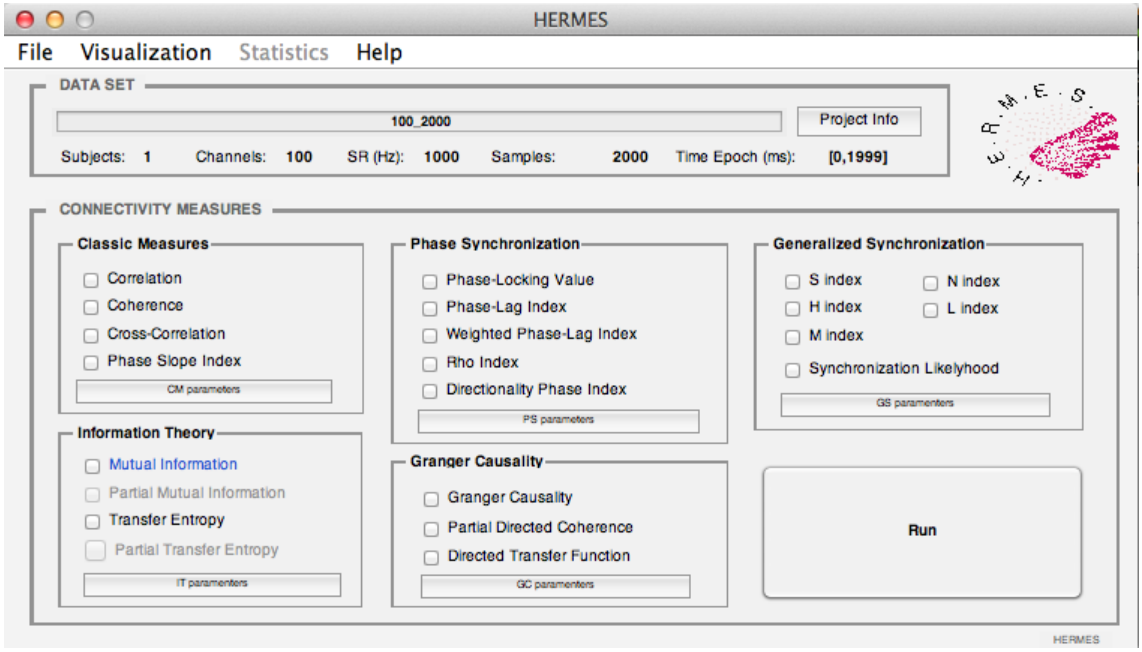

Figure 1.1: HERMES Graphic User Interface (GUI).

The main pourpose of HERMES is the analysis of brain FC and EC. Therefore, it does not include any artefact-removal, detrending or any similar pre-processing tools.

HERMES includes several types of connectivity indexes. Although the indexes included in HERMES can be classified in to two main groups: FC indexes, which measure statistical dependence between signals without providing any causal information, and EC indexes, which do provide such causal information; HERMES classifies them in five different categories: *clasical* measures, *Phase synchronization (PS)* indexes, *Generalized synchronization (GS)* indexes, *Granger casuality* based indexes and *information theoretic* indexes.

1.2 Goals

- Optimise the information theoretic estimators functions used by HERMES
  Identified tasks:
  - Become familiar with HERMES tool.
  - Analyse TIM functions used by Hermes.
  - Measure time used by HERMES to calculate MI and TE with different scenarios.
  - Propose and implement solutions to improve the time.
  - Analyse the proposed solutions.
- Allow the new algorithms to be used by HERMES in different operative systems.

1.3 Structure

In chapter 2 definitions about information theory estimators used by HERMES are presented. The principal goal of this project is the optimisation of the functions used to calculate this estimators.

The libraries used to calculate the estimators by HERMES tool, are introduced in chapter 3. A preliminary analysis of the use of these libraries from HERMES to calculate the estimators are presented in chapter 4.

The work made to achieve the goal, that consists on the optimised the process to calculate the estimators, the creation of the HermesTim library and the MATLAB interface, together preliminar results and a OpenMP analysis to find improvements to apply to the new library is shown in chapter 5. The measures obtained with the new library in the different scenarios are presented in chapter 6.

Finally, the conclusions and future work are shown in chapter 7.

In addition an appendix, with the installation and compilation manual (appendix A) are added to the document.

Chapter 2

Data Analysis

2.1 Shannon Entropy

In information theory, entropy is a measure of the uncertainty in a random variable[12]. Shannon entropy quantifies the expected value of the information contained in a message. Shannon entropy is the average unpredictability in a random variable, which is equivalent to its information content. The concept was introduced by Claude E. Shannon in his 1948 paper *A Mathematical Theory of Communication*[22].

Suppose that we have a set $a_1, \ldots, a_{M_a}$ of possible states whose probabilities of occurrence are given by $(p_1, \ldots, p_{M_a})$. Then the random experiment is described by a random variable X with probability

$$P(X = a_i) = p_i,\ i = 1, \ldots, m. \tag{2.1}$$

The $(p_1, \ldots, p_{M_a})$ is the probability distribution of X.

Then the entropy of the random variable X or the entropy of the distribution $(p_1, \ldots, p_{M_a})$ is defined by

$$H(A) = -\sum_{i=1}^{M_A} p_i \log p_i \tag{2.2}$$

Suppose that a probability distribution $(p_1, \ldots, p_{M_a})$ is known and that we do not know which event will occur. Then the entropy $H(p_1, \ldots, p_{M_a})$ shows how much freedom one is given in the selection of an event, or how uncertain the outcome is or how difficult to predict the outcome.

The logarithm may be taken to the base e or to the base 2. In the case of base e, the entropy is measured in units of **nats**, while in the case of base 2, the entropy is measured in units of **bits**. Bit is usually more convenient for the practical use, and nat is more convenient for theoretical developments.

Some properties of $H(A)$ are:

- If $p_l = 1$ and all other probabilities $p_i = 0$ with $i \neq l$ then $H(A) = 0$.

- For equiprobable events the entropy $H(A)$ is maximal.

$$p_i = \frac{1}{M_a} \forall i \Longrightarrow H(A) = \log M_a \tag{2.3}$$

- If probability of event $M_{a+1} = 0$, and is added to system A, then the entropy remain unchanged.

$$H(A) = H(p_1, \ldots, p_{M_a}, p_{M_{a+1}}) = H(p_1, \ldots, p_{M_a}, 0) \tag{2.4}$$

- If the logarithm to base M_a is used, the entropy is normalised.

$$0 \leq H(A) \leq 1 \tag{2.5}$$

The joint entropy $H(X, Y)$ of two discrete variables X and Y is defined analogously

$$H(X, Y) = -\sum_{i=1}^{M_x} \sum_{j=1}^{M_y} p(x_i, y_j) \log p(x_i, y_j) \tag{2.6}$$

Here $p(x_i, y_j)$ denotes the joint probability that X is x_i and Y is y_j. The number of possible values M_x and M_y may be different. If X and Y are statically independent the joint probabilities factorise and the joint entropy $H(X, Y)$ becomes

$$H(X, Y) = H(X) + H(Y) \tag{2.7}$$

In general, however, the joint entropy may be expressed in terms of the conditional entropy $H(X|Y)$

$$H(X, Y) = H(X|Y) + H(Y) \tag{2.8}$$

with $H(X|Y)$ being defined as

$$H(X|Y) = -\sum_{i=1}^{M_x} \sum_{j=1}^{M_y} p(x_i, y_j) \log p(x_i|y_j) \tag{2.9}$$

Since for arbitrary random variable X and Y

$$H(X|Y) \leq H(X) \tag{2.10}$$

we get the relation

$$H(X, Y) = H(X) + H(Y) \tag{2.11}$$

instead equation 2.7.

2.2 Mutual Information

In probability theory and information theory, the MI of two random variables is a quantity that measures the mutual dependence between them.

Formally, the mutual information of two discrete random variables X and Y can be defined as:

$$I(X,Y) = \sum_{i=1}^{M_x} \sum_{j=1}^{M_y} p(x_i, y_j) \log \left(\frac{p(x_i, y_j)}{p(x_i)p(y_j)} \right) \tag{2.12}$$

where $p(x, y)$ is the joint probability distribution function of X and Y, and $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y respectively.

Intuitively, MI measures the information that X and Y shares: it measures how much knowing one of these variables reduces uncertainty about the other. For example, if X and Y are independent, then knowing X does not give any information about Y and vice versa, so their mutual information is zero. But, if X and Y are identical then all information carried by X is shared with Y, knowing X determines Y. In the case of identity the mutual information is the same as the uncertainty contained in X alone, that is the entropy of X.

MI is a measure of the inherent dependence expressed in the joint distribution of X and Y relative to the joint distribution of X and Y under the assumption of independence. Among the measures of independence between random variables, MI is singled out by its information theoretic background[6]. In contrast to the linear correlation coefficient, it is sensitive also to dependencies which do not manifest themselves in the covariance. MI is zero if and only if the two random variables are strictly independent.

The mutual information $I(X,Y)$ between two random variables X and Y is defined as [22, 13]

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{2.13}$$

and applying equation 2.8

$$I(X,Y) = H(X) + H(Y) - H(X,Y) \tag{2.14}$$

2.2.1 Mutual Information Estimation

Up to now all the definitions of MI implies previously knowing the probability distributions of each random variable, but in general they are not know. It is possible to estimate this from experiment measurements. The two methods that are explained later follow this concept, but more techniques are used to estimate the MI. Kraskov[14] suggested a technique based in the use a Kozachenko-Leonenko[15] estimator for Shannon entropy's to calculate one variable entropy's and a his own entropy estimator based in adaptive partitioning to calculate multivariate entropy's.

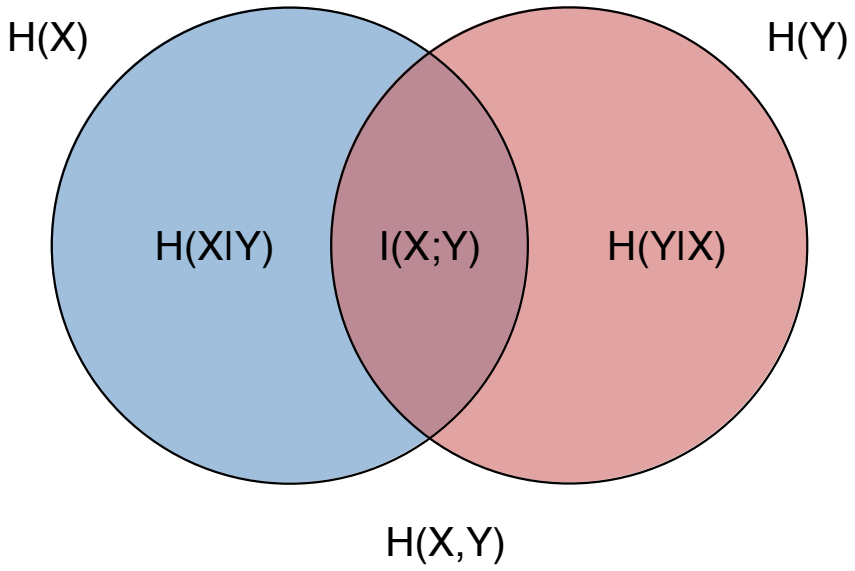

Figure 2.1: Mutual information and entropy relationship.

Histogram-based methods

Considering a collection of N simultaneous measurements of two continuous variables x and y.

One of the most straightforward approaches is, to use a histogram based technique. Given an origin o and a width h, the bins of the histogram for the variable x are defined through the intervals $[o + mh, o + (m + 1)h]$ with $m = 0, \ldots, M$. The data are thus partitioned into M discrete bins a_i and k_i denotes the number of measurements that lie within the bin a_i. The probabilities $p(a_i)$ are then approximated by the corresponding relative frequencies of occurrence

$$p(a_i) \to \frac{k_i}{N} \tag{2.15}$$

and the mutual information $I(X, Y)$ between both datasets X and Y may be expressed as

$$I(X, Y) = \log N + \frac{1}{N} \sum_{ij} k_{ij} \log \frac{k_i j}{k_i k_j} \tag{2.16}$$

Here $k_i j$ denotes the number of measurements where x lies in a_i and y in b_j.

It is known that the estimation of entropy's from finite samples may be affected by systematic error [9, 10].

$$\left\langle H(X, Y)^{observed} \right\rangle \approx H - \frac{M - 1}{2N} \tag{2.17}$$

Here $H^{observed}$ denotes the estimated entropy using a finite sample of N datapoints to estimate the probabilities of M discrete states. It should be pointed out that in this approximation the systematic error is independent of the underlying probability distribution. As mutual information can be defined as a sum of entropy, it is possible to use this expression to estimate the systematic error of $I(X,Y)$[11].

$$\left\langle I(X,Y)^{observed} \right\rangle \approx I(X,Y)^{true} + \Delta I(X,Y) \tag{2.18}$$

with

$$\Delta I(X,Y) = \frac{M_x y - M_x - M_y + 1}{2N} \tag{2.19}$$

Here M_x, M_y and $M_x y$ denote the number of discrete states (histogram bins) with non zero probability.

Other approach is based in the adaptive partitioning. Mutual information depends on the distribution of the individual datasets[], $I(X,Y)$ is bounded by the individual entropy's of X and Y.

$$I(X,Y) \leq \min H(X), H(Y) \tag{2.20}$$

To be sure that results can be compared between them it is necessary that they are not blurred by the individual distributions of each dataset values. One of the most straightforward strategies is to normalise all measured datasets to and identical reference distribution. This can be done using an adaptive partitioning method to divide the axes in bins. With this method, each axis is partitioned into M discrete bins, each bin approximately containing $k = N/M$ datapoints. Consequently, the width of each interval is determined by the local density of the measured dataset.

Kernel Density Estimation (KDE)

Other method to estimate the MI, suggested by Moon[18], is based on Kernel Density Information (KDE). This method was found to be superior in terms of:

- Better mean square error rate of convergence of the estimate of the underlying density.
- Insensitivity to the choice of origin.
- Ability to specify more sophisticated window shapes than the rectangular window for frequency counting[18, 2].

For better comprehension about KDE read [2]. This method aims to improve the probability density estimate $p(x)$. It is applicable in many other situations. The first step is free the histogram from a particular choice of origin and bin positions. This results to in the naive estimator

$$\hat{f}(x) = \frac{1}{2Nh} \sum_{i=1}^{N} \Theta(h - \|x - x_i\|) \tag{2.21}$$

where $\Theta(x)$ denotes the Heaviside function

$$\Theta(x) = \begin{cases} 0 & six > 0 \\ 1 & six \leq 0 \end{cases} \tag{2.22}$$

A graphical interpretation of equation 2.21 is that the estimator is obtained by additively putting boxes of width $2h$ and height $(2Nh)^{-1}$ on each observation. Other shapes still lead to a valid estimate of the probability density. With a generalised weight or kernel function $K(x)$ the KDE $\hat{f}(x)$ is given by

$$\hat{f}(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right) \tag{2.23}$$

The parameter h is called *window width* and the kernel function $K(x)$ is required to be probability density. A possible kernel could be the Gaussian kernel, in this case the Gaussian estimator may be explained as placing Gaussian 'bumps' at the position of each observation of x_i. It is very important the choice of the bandwidth h, if h is chosen to small spurious fine structure becomes visible, while if h is too large all detail, spurious or otherwise is obscured. Different methods to chose an appropriate bandwidth available exists, but most of thenm imply a lot of computation time[18].

The mutual information $I(X, Y)$ is a function of probability densities. Thus an obvious way to find an estimate for $I(X, Y)$ is to find estimates of the densities and then substitute these into the required integral.

$$\hat{I}(X, Y) = \int_x \int_y \hat{f}(X, Y) \log \frac{\hat{f}(x, y)}{\hat{f}(x)\hat{f}(y)} dxdy \tag{2.24}$$

2.3 Transfer Entropy

TE is a non-parametric statistic measuring the amount of directed (time asymmetric) transfer of information between two random variables[21]. TE from a random variable X to another random variable Y is the amount of uncertainty reduced in future values of Y by knowing the past values of X given past values of Y.

Supposing two systems which generate events, the *entropy rate*, which is the amount of additional information required to represent the value of the next observation of one of the system is defined as

$$h_1 = -\sum_{x_{n+1}} p(x_{n+1}, x_n, y_n) \log p(x_{n+1}|x_n, y_n) \tag{2.25}$$

Supposing that the value of observation X_{n+1} is independent of the current observation y_n

$$h_2 = -\sum_{x_{n+1}} p(x_{n+1}, x_n, y_n) \log p(x_{n+1}|x_n) \tag{2.26}$$

The quantity h_1 represents the entropy rate for the two systems, and h_2 represents the entropy rate assuming that X_{n+1} is independent of Y_n. Thus, the transfer entropy $T_{Y\to X}$ is

$$\begin{aligned} h_2 - h_1 &= -\sum_{x_{n+1}} p(x_{n+1}, x_n, y_n) \log p(x_{n+1}|x_n) \\ &\quad + \sum_{x_{n+1}} p(x_{n+1}, x_n, y_n) \log p(x_{n+1}|x_n, y_n) \\ &= \sum_{x_{n+1}} p(x_{n+1}, x_n, y_n) \log \frac{p(x_{n+1}|x_n, y_n)}{p(x_{n+1}|x_n)} \end{aligned} \tag{2.27}$$

It can be written as

$$T_{Y\to X} = H(X^{t+1}|Y^t) - H(X^{t+1}|X^t, Y^t) \tag{2.28}$$

Transfer entropy has been used for estimation of functional connectivity of neurons in HERMES.

Chapter 3

Libraries

Hitherto HERMES has used information theory estimators given by TIM[1] library. TIM relies partially on the Pastel[7] library. These two libraries have been developed by Kalle Rutanen and they are under open source license.

HERMES is a application developed in MATLAB. Therefore, an introduction of MATLAB and C/C++ programming language integration with MATLAB is given in section 3.3.

3.1 Pastel

Pastel is a cross-platform C++ library for geometry and computer graphics, it is parallelised with OpenMP. This library is under continuous improvement and it undergoes several changes between versions.

This library is divided into seven sub-libraries:

- PastelGeometry: Library of geometry algorithms and data structures. A unifying principle across this sub-library is that geometric problems are solved independent of dimension, whenever that is possible.
- PastelGfx: Library that provides algorithms related to computer graphics.
- PastelGfxUi: Library that provides a simple graphical user interface. This library is dependant on both PastelGfx and PastelDevice, which are independent of each other.
- PastelMath: Library that provides general-purpose mathematical tools. Some of the basic mathematical tools can be found from PastelSys sub-library.
- PastelDevice: Library that provides tools for accessing hardware. It is a wrap around the Simple DirectMedia Layer (SDL) library.
- PastelDsp: Library that provide tools for digital processing. These tools are related to re-sampling, filtering, and transforming between time and frequency domains.

- PastelGl: Library that provides tools that rely on OpenGl.
- PastelSys: Library that provides general-purpose tools needed in almost any non-trivial program.

And has two wrapper libraries to be used from MATLAB:

- PastelGeometryMatlab: Library that provides a MATLAB interface to the data structures and algorithms of the PastelGeometry library.
- PastelMatlab: Library that provides tools for easier interfacing with MATLAB, when creating mex files (see section 3.3.1). It provides the mex entry point, convenience functions for retrieving MATLAB arguments, and a way to register multiple functions to be callable via the entry point.

Pastel is distributed under MIT license, and it can be found in `http://kaba.hilvi.org/homepage/`.

3.2 TIM

TIM is a library for efficient estimation of information-theoretic measures from continuous-valued time-series in arbitrary dimensions. It is developed in C++, and it is cross-platform. TIM provides a MATLAB interface that allows it to be used from MATLAB, as well as a console interface.

TIM allows to estimate Shannon's differential entropy using a diversity of estimators, and other entropy's: Renyi, Tsallis, etc. Moreover, TIM offers methods to calculate entropy combinations like mutual information, partial mutual information, transfer entropy and partial transfer entropy.

TIM consists of two sub-libraries:

- TimCore: Library that provides the estimation methods.
- TimMatlab: Wrap of TimCore to be used from MATLAB.

And a console application. The console application receives MATLAB-based scripts as input and gives the results. This script includes the data and the operations to compute.

TimCore is the main part of the library as it encompasses all the functions implemented as part of the library.

`Signal` and `SignalPointset` classes are the fundamental pillar of TimCore library. The first models a time-series as representing it as a matrix of values and allowing it to be manipulated correspondingly. The second class in turn models a reinterpretation of a time-series as a semi-dynamic set of points.

TIM relies on the PastelSys sub-library of the Pastel library (section 3.1), using data structures and types defined in PastelSys.

3.2.1 Iterators

TIM uses iterator ranges to abstract away the differences between different kinds of containers. The iterators are based in iterator software design pattern, in which an iterator is used to traverse a container and access the container's elements. The iterator pattern decouples algorithms from containers.

An iterator range is a triple (b, e, d) where b and e are the begining and end of the iterator and make up the iterator range, d is the distance between b and d.

3.2.2 Generic entropy estimation

The generic entropy estimator refers to an algorithmic skeleton which is used to compute de k-nearest-neighbour-based entropy estimators. The algorithms for the estimation of Renyi entropy, Tsallis entropy, and Shannon differential entropy share a very similar estimation algorithm, with the differences being localised to a few points. The generic entropy estimator encapsulates this similarity and allows to customise these key points.

3.3 MATLAB

MATrix LABoratory (MATLAB) is a numerical software tool that provides an integrated development environment (IDE) and uses M programming language. MATLAB is developed and supported by MatWorks[1]. It offers high speed development because of its high level programming, but it is a proprietary software with an annual fee. This software is multi-platform and can be executed in Linux, OS X and Windows.

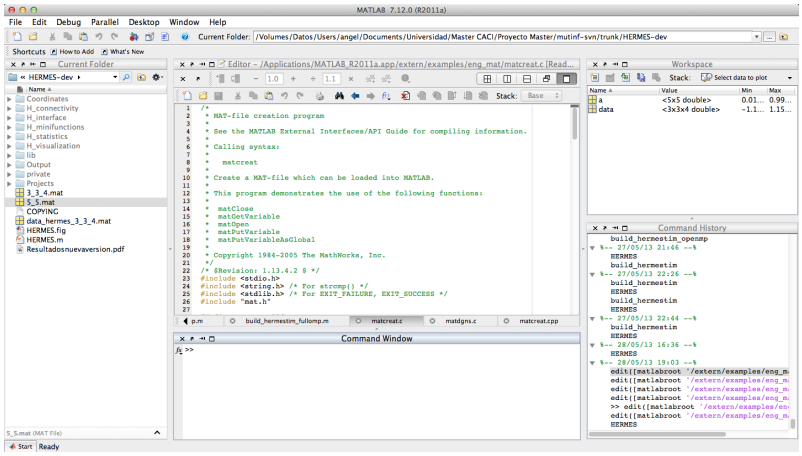

Figure 3.1: MATLAB integrated development environment (IDE).

MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of GUI and interfacing with programs written in

[1] `http://www.mathworks.com`

other languages, including C, C++, Java and FORTRAN, and communications with hardware devices. The functionality of MATLAB could be incremented using toolboxes, MathWorks provides different toolboxes oriented to many fields: statistics, test and measures, computational biology, etc, though almost all are paid products, it is possible to get third-party toolboxes. The work described in this document is related to the HERMES toolbox which is a public toolbox.

MATLAB is a matrix and vectors numerical software with an interpreted language. This means that M functions or scripts are not compiled before execution, this has relevant performance losses, although almost all built-in functions are developed in other languages, normally in C, and they have a high optimisation level. The possibility of using other languages to do things in MATLAB is also offered to the end-user, being able to use functions and subroutines written in C, C++ and FORTRAN with better performance. For this purpose it is neccesay to make a wrap of the function, this procedure is explained in the next section (3.3.1).

3.3.1 MEX: MATLAB interface with other programming languages

MATLAB gives the possibility to use functions and subroutines written in other programming languages like if they were functions written in the M language. This way, it is not necessary to rewrite the functions in M language, taking advantage of the underlying benefits: high performance and re-usability of code, particularly using `for-loop` sentences where MATLAB is quite slow. This gives us the opportunity to use other technological benefits like multi-thread parallelisation given by *OpenMP*[2][20].

In this section we explain the way to use C/C++ functions and subroutines, although the process is similar with other languages.

This feature is achieved implementing a gateway routine, that must be named `mexFunction`, that interfaces between MATLAB data types and C/C++ data types, and calls the computational routine/s. The computational routine can be part of other libraries or functions, or it can be implemented within the gateway function. The gateway routine receives four parameters:

- `prhs`: A vector with input arguments.
- `plhs`: A vector with output arguments.
- `nrhs`: Number of input argument, size of `prhs` vector.
- `nlhs`: Number of output arguments, size of `plhs` vector.

The gateway function skeleton, `mexFunction`, is shown in the listing 3.1

[2] `http://openmp.org`

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
    /* C implementation */
}
```

Listing 3.1: MEX gateway function skeleton.

The file containing the gateway routine implementation is called a MEX file, which really is an external language file that includes the MATLAB entry point and it is compiled with the MEX system included with MATLAB.

MEX files can be called as MATLAB functions, but first, MEX files must be compiled to be used from MATLAB. The MATLAB function name associated with the MEX file is the same as the name of the MEX file. The arguments of the MATLAB function will be pass trough the MEX function. When the MEX function is called, it start executing the gateway function. The gateway function should implement the computational routine or call functions or a routine that implement the computational routine.

The compiled MEX files are platform dependant, they receive a different extension depending on the underlying platform, because of this it is necessary to compile the MEX file in each platform where we want to use it. Once the MEX files are generated for each platform, they can be distributed in binary format and do not need to be compiled again. By using the MATLAB function `mexext` we can know the extension in the underlying platform.

MEX files require the declaration of at least the *mex.h* header file. In this header file the MATLAB data types to be used in C/C++ language and declares auxiliary functions to create, destroy, etc this types are defined.

The basic type of the MATLAB C/C++ interface is `mxArray`, which is an abstract type that encapsulate other types. Scalars, vectors and matrix declared in MATLAB are represented as `mxArray` in C MEX file. The MATLAB interface provides functions that allow users to know the real type of a `mxArray`, conversions between `mxArray` and C types and create, destroy, etc. Some definitions of auxiliary functions are showed in listing 3.2. The `matrix.h` header file must be included to use `mxArray` within C/C++ MEX files.

```
mxCreateLogicalScalar // Create an scalar
mxCreateCharArray     // Create an string
mxGetPr               // Return C pointer to C vector containing data elements of
                      // the Matrix given as argument.
```

Listing 3.2: Examples functions of MATLAB API.

Knowing how data types of external languages are stored in memory is a key point when complex data structures are managed. MATLAB complex data structures are *Arrays*, *Matrix* and *Cells*, these types can be passed as arguments to external languages functions and it is neccesary to know how to manipulate these. An example, MATLAB store matrix elements by columns, as showed in figure 3.2, but matrix elements are stored normally by rows in C/C++.

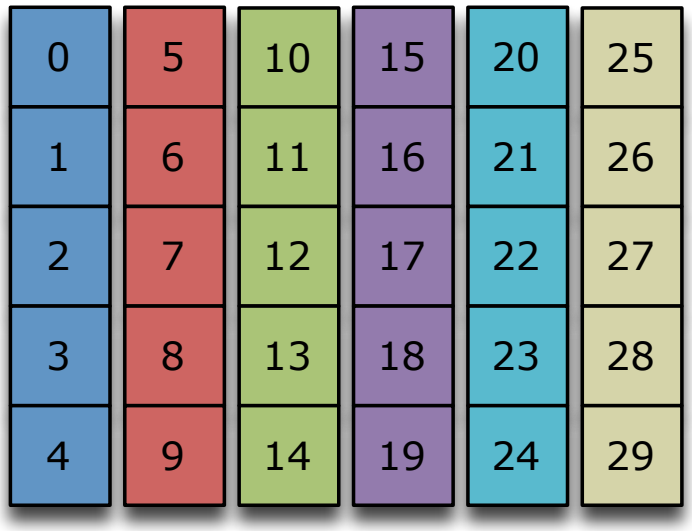

Figure 3.2: MATLAB matrix elements position in memory.

3.3.2 MEX files compilation

MATLAB includes the MEX compilation system. A MEX function can not be used if it is not compiled previously. The compilation generates a binary code library that can be used from MATLAB as it was an M function.

The MEX compilation system is based in the standard compiler of the underlying platform: GCC in Linux, Visual Studio in Windows, etc, although in the Windows version a internal C compiler, LCC, is distributed with MATLAB. A complete list of compatible compilers can be found in the MatWorks web page[3]. Before compiling any MEX file it is necessary to indicate the compiler to be used, this is achieved by executing `mex -setup` in the MATLAB console.

When the MEX compiler system has been configured, the mex files are compiled using the `mex` command followed by the name of the MEX file. An example is showed in listing 3.3.

```
1 mex examplemex.c
```

Listing 3.3: MEX compilation example command.

[3] `http://www.mathworks.es/support/compilers/R2013a/index.html`

Chapter 4

Previous Analysis

In this section the previous analysis made about the tools and libraries used by HERMES to calculate the information theory estimators until now is explained. The analysis mainly focuses on the function that calculates the mutual information.

Before starting it is necessary to understand some concepts: the first is the data sets used by HERMES, that are real MEG records, and are formed by a number of *samples* for each *channel*, or sensor, used in MEG. Normally, this type of experiments are repeated more than one time, and each record with all channels and samples is called *trial*. A dataset with *trials* is represented as a three-dimensional matrix, each record is a "samples x channel" matrix. Figure 4.1 shows a memory map of a dataset in a three-dimensional matrix such as likeMATLAB assigns it to memory. The elements values match with the index of the element since all the elements in the matrix are contiguous in memory. This assignment in memory of the datasets is taking into account throughout the development process giving us an improvement in memory use, avoiding conversion between MATLAB and C types.

The second concept is about the method used to measure the parallelisation level. In parallel computing, *Speedup* refers to how much a parallel algorithm is faster than a corresponding sequential algorithm. And it is defined by the equation 5.4. In the analysis this concept is used to compare results.

$$S_p = \frac{T_1}{T_p} \tag{4.1}$$

Where:

- p is the number of processors.
- T_1 is the execution time of the sequential version.
- T_p is the execution time of the parallel version.

The ideal speedup is obtained when $S_p = p$.

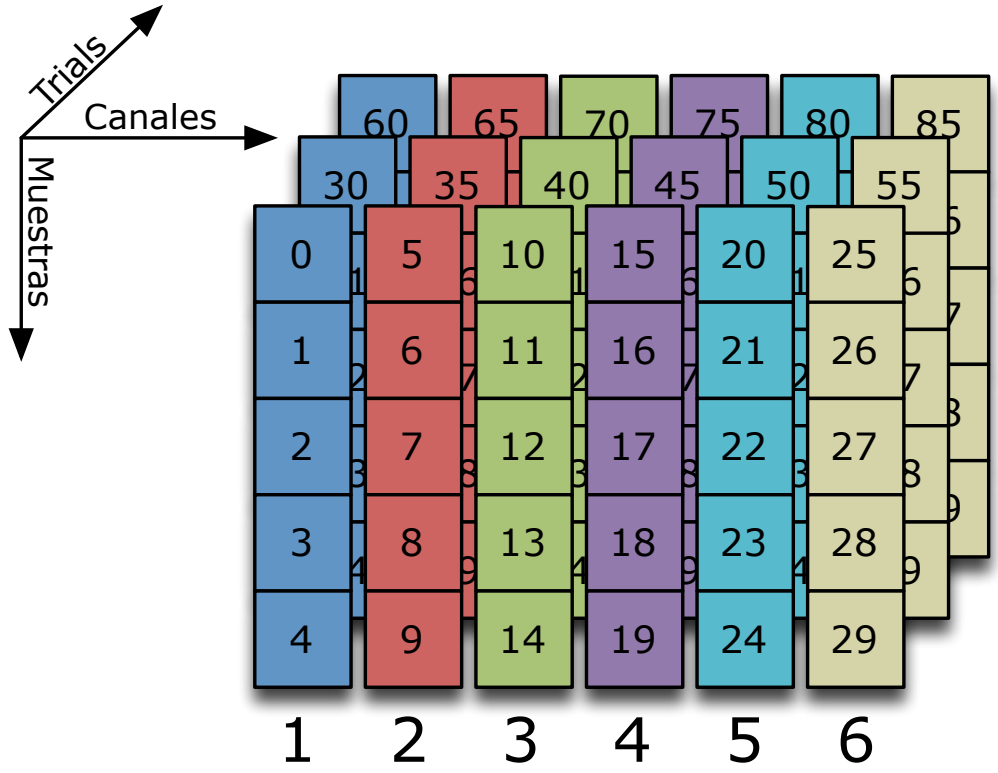

Figure 4.1: Dataset with trials memory map. Represented as a 3-d matrix.

4.1 Temporal Analysis

As a previous step to localise problems and possible regions to optimise it is necessary to get reference times. The same compilation flags are used in the development process to be able to compare the measured times. This flags have influence in the compiler optimisation. If different compilation flags are used the measured times could give us a wrong idea. These flags will be used to compile all the libraries used in the analysis and development stages of this work.

HERMES uses TIM and Pastel library to compute mutual information and transfer entropy for signal sets, and the corresponding software architecture is showed in 4.2. The first step of this work is to compile these libraries with the compilation flags used in the development process:

- -O2: Optimisation level 2, according to GCC Manual[8], the compiler performs nearly all supported optimisation's that do not involve a space-speed tradeoff.
- -g: Produce debugging information.

The measures are doing over three different scenarios, each with a different dataset size:

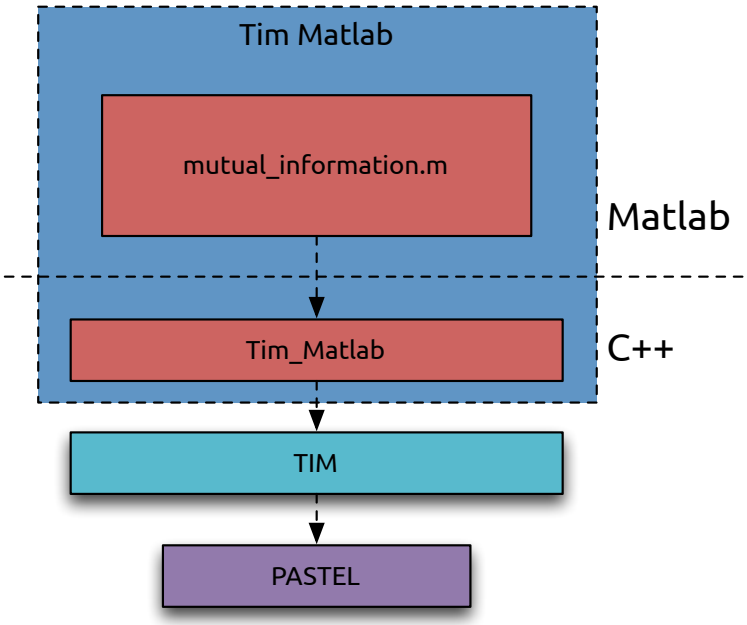

Figure 4.2: Tim_Matlab Software Architecture

Name	Processor	N Processors	N Cores	Memory RAM	Version GCC
Ebano	AMD FX-8350	1	8	16GB	4.6
Espino	Xeon 5500	2	12	48GB	4.4
Magnolio	Opteron	4	48	16BG	4.4
MacBook	Core2Duo P8600	1	2	8GB	4.7

Table 4.1: Features summary of computers used.

- Scenario 1: Two seconds of real MEG with a frequency of 1000Hz. 100 channels and 2000 samples per channel.

- Scenario 2: Five seconds of real MEG with a frequency of 1000Hz. 150 channels and 5000 samples per channel.

- Scenario 3: Two seconds of real MEG with a frequency of 1000Hz, analysis repeated 30 times. 100 channels and 2000 samples per channel, with a total of 30 *trials*.

The measures are done in four different computers, each computer has a tag name associated and are presented in table 4.1.

The tests consist in getting the execution time used to estimate the mutual information for each channel pair. Having c channels, or signals, and knowing that $I(A, B) = I(B, A)$, it is not necessary to calculate all the pairs, this gives a p number

of calls to mutual information function, shown in equation 4.2.

$$p = \binom{c}{2} \tag{4.2}$$

	Scenario	Sequential Time	Parallel Time	Speedup
MacBook	Scenario 1	168	129	1.30
	Scenario 2	985	762	1.29
Ebano	Scenario 1	96	70	1.37
	Scenario 2	653	378	1.73
	Scenario 3	11700	3174	3.69
Espino	Scenario 1	102	81	1.26
	Scenario 2	682	323	2.11
	Scenario 3	12840	2195	5.85
Magnolio	Scenario 1	124	73	1.70
	Scenario 2	848	365	2.32
	Scenario 3	21480	2409	8.92

Table 4.2: Times used for each computer and obtained speedup.

The measured times and the corresponding speedups are showed in table 4.2. All the Speedups are very low in all the test, only the SpeedUp in scenarios with *trials* is a little better but still remains a long way from the ideal speedup of each computer. Because of the low SpeedUp obtained in the tests, we decide to do a dynamic study to be able to understand the problems.

4.2 Dynamic Analysis, *Profiling*

The Valgrind tool[1] is used to do the dynamic analysis. Valgrind creates a *virtual* execution environment where a native application runs. When an application is running in the virtual environment Valgrind registers the execution times of each function, the number of times that each function is called, etc. Because of the characteristics of Valgrind it is necessary to have a native application that executes the process to analyse. If we run HERMES in Valgrind, the analysis is done to the MATLAB environment, because HERMES is a toolbox of MATLAB, and not to the process to analyse.

To that purpose we choose to develop a C++ application that does a very similar process as HERMES to calculate the mutual information, which includes the use

[1] http://valgrind.org

of the mutual information estimator offered by the TIM library. The development process of this application was harder, Pastel and TIM libraries compilation was more laborious and arduous process that was estimated at first moment. The next step was to come to understand how to use the *mutual information* function given by the TIM library from the C++ program. Although TIM is developed in C++, the lack of documentation make it harder to use. The header of the mutual information function defined by TIM is listed in 4.1.

```
.......
    //! Computes mutual information.
        /*!
        Preconditions:
        kNearest > 0
        ySignalSet.size() == xSignalSet.size()

        xSignalSet, ySignalSet:
        A set of measurements (trials) of
        signals X and Y, respectively.

        xLag, yLag:
        The delays in samples that are applied to
        signals X and Y, respectively.

        kNearest:
        The number of nearest neighbours to use in the estimation.

        If the number of samples varies between trials,
        then the minimum number of samples among the trials
        is used.
        */

        template <
               typename SignalPtr_X_Iterator,
               typename SignalPtr_Y_Iterator>
        real mutualInformation(
               const ForwardIterator_Range<SignalPtr_X_Iterator>& xSignalSet,
               const ForwardIterator_Range<SignalPtr_Y_Iterator>& ySignalSet,
               integer xLag = 0, integer yLag = 0,
               integer kNearest = 1);
        ......
```

Listing 4.1: Tim mutual_information definition.

The MI function offered by TIM requires two parameters, these are an iterator range (see 3.2.1) for signal X and Y. TIM library defines a signal as a set of variables of a `real` type contiguous in memory. The `real` type is defined by TIM as a configurable type, `double` by default.

Taking into account the definition in the previous paragraph, we have created a program that loads from text files a set of c signals with n samples each one, and stores the elements of equal signals contiguous in memory. Different signals do not need to be contiguous in memory (figure 4.3).

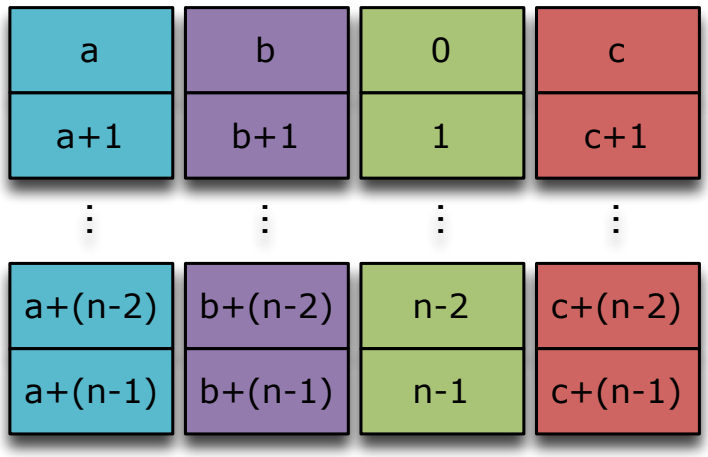

Figure 4.3: Relationship between signals and memory.

To get the iterator range of the signal, we create an iterable data structure with a pointer to each trial signal previously created, and then we obtain the beginning and end iterator of the collection that are passed as arguments to the `range` function of the BOOST[2] library.

Once the application that uses TIM library to estimate the mutual information between two signals was developed, we started the dynamic analysis of it. The application read the data from a file and calls TIM *mutual information* only one time for each signal pair, having a total of p calls (see equation 4.2).

To make the analysis, the application is compiled with internal parallelisation of TIM activated. The profiling was done with Valgrind using a dataset with 50 channels and 1999 samples for each. The dataset used for this test was smaller because the overhead introduced by Valgrind is around $10 \sim 15$ times of the corresponding sequential program.

In figure 4.4 the results obtained from Valgrind are showed. We have used a graphical applications called QCacheGrind[3] that help us to understand the results given by Valgrind in a easier way.

From the results, we can conclude that only one function makes enough work to be analysed, around 30%. The other functions use no more than 4% of total time being called thousand of times. The function that is analysed is `Search_Nearest_Neighbours`, it is included in the Pastel library and calculates the nearest neighbours for a given point. From the analysis done to `Search_Nearest_Neighbours`, we conclude that the function was well optimised.

[2] http://www.boost.org
[3] `http://kcachegrind.sourceforge.net/html/Home.html`

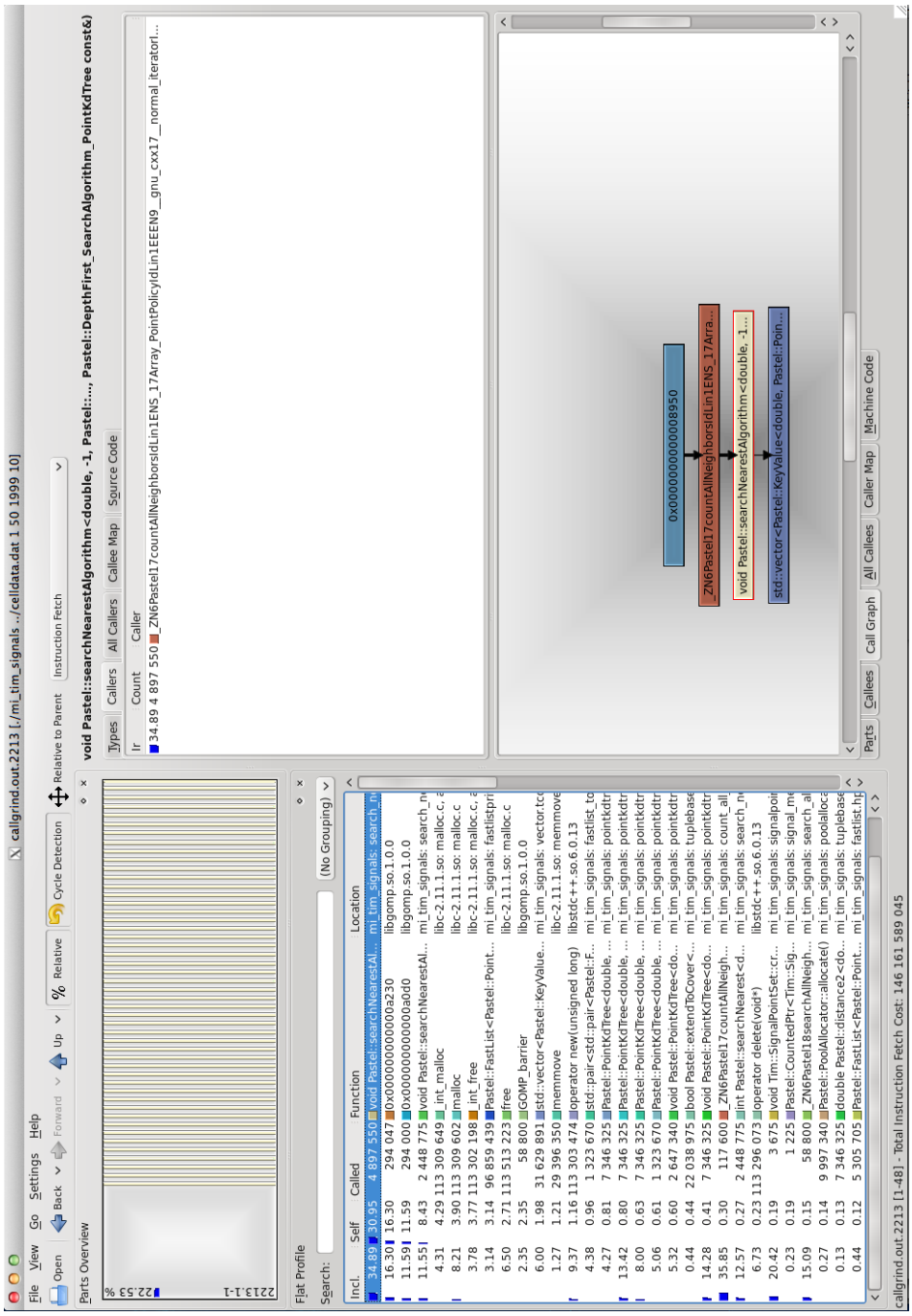

Figure 4.4: Profiling results showed with KCacheGrind tool.

Chapter 5

Development

5.1 HermesTim: Library Parallelisation

Due to the little margin to optimise the Pastel and the TIM library we choose to analyse why the speedup with the parallelised version of the original libraries is so low. We conclude that the low Speedup could be produced because the parallel regions have a small workload, this is called fine-grained parallelisation. In case that this assumption was true, the process to assign work for each thread in parallel regions produces more overhead that the time won parallelising the work.

In HERMES, the mutual information and entropy combination is always calculated for a signal set, normally about one hundred signals or more. We think that parallelising the application with a bigger workload for each thread, called coarse-grain parallelisation, gives us a better speedup improvement owing lower overhead produced by the work scheduler. We define as new workload the process of computing the routine (mutual information or transfer entropy) for each channel pair, this is implemented calling the Tim library function for each channel pair, in total p calls to the corresponding function are done, see equation 4.2. For example, for 100 channels we have 4950 work units for mutual information, and 10000 work units for transfer entropy, that is enough work to be divided among all threads.

This part of the work consists in creating a new software library, named HermesTim, that mediates between the HERMES toolbox and the TIM library (figure 5.1). To be used from HERMES another software layer on top of it, that interfaces MATLAB with C++ is needed. HermesTim library is intended to contain parallelised wrappers to the TIM functions used in HERMES: mutual information and transfer entropy.

A wrapper for each one is implemented as a result of this work. The optimisations described in this work and measures of results are made using mutual information function as a base, owing that the wrappers are very similar, though some measures are made over transfer entropy function.

This new library does not work with signal pairs as the TIM library works, but it works with signal sets doing the operations over all the signals. Because of this, the functions declared in HermesTim define a dataset with all the signals as

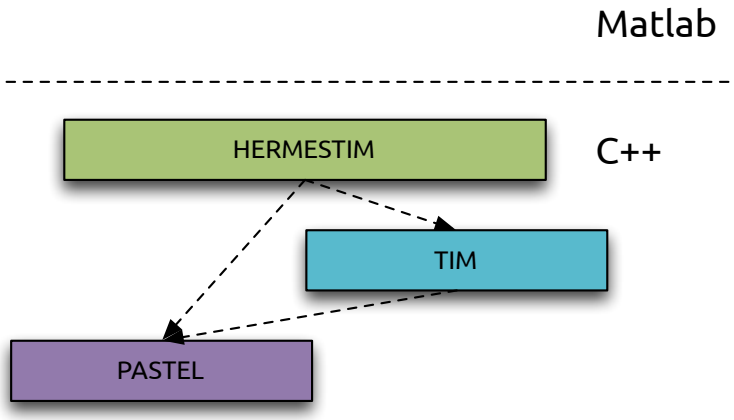

Figure 5.1: HermesTim Software Architecture

input parameters. The format of this data is consistent with the data structure defined at the beginning of section 4, see figure 4.1. The use of this structure gives a performance improvement when the library is used from MATLAB because no conversion or remapping of data must be done.

5.1.1 Mutual Information

The first wrapper implemented in HermesTim computes the mutual information for a set of signals. This function uses the mutual information estimator offered by the Tim library to compute the mutual information for each pair of signals. Due to mutual information theory,

$$I(X,Y) = I(Y,X) \tag{5.1}$$

the number of mutual signal pairs to compute is less than c^2, where c is the number of signals in the dataset.

The header definition of the HermesTim mutual information function is showed in 5.1. In this case, in addition to the signal dataset pointer, the input parameters declared are: a result matrix pointer; the number of samples, channels and trials of the input datasets; xLag, yLag and the number of neighbours. The three last parameters are function dependant, these parameters are needed by Tim's mutual information as we can see in TIM mutual information definition (listing 4.1).

```
namespace HERMESTim {
    void mutual_information(double* data, double *dataMI,
                            int samples, int channels, int trials,
                            int xLag, int yLag, int K);
}
```

Listing 5.1: HermesTim mutual_information definition

To achieve the parallelisation with the workload defined previously the OpenMP directive *parallel for*[20] has been used. This OpenMP directive automatically parallelises the underlying C++ *for* block, being each loop iteration a work unit that is scheduled at run-time by the OpenMP library. The OpenMP scheduler shares out the iterations between the threads available depending on the OpenMP schedule policy used. The OpenMP schedule policy can be modified using environment variables or using OpenMP's application programming interface (API). The source code of parallel region implemented is shown in listing 5.2.

```
.......
        for(int i = 0; i < channels; ++i)
        {
            outData.insert(i,i,1); // MI of the same channel

            #pragma omp parallel for default(none) shared(cells, outData,
            channels, xLag, yLag, K, i) schedule(runtime)
            for (int j = i + 1; j < channels; ++j){
                std::vector<Tim::SignalPtr> *xCell; // Cell for x channel
                std::vector<Tim::SignalPtr> *yCell; // Cell for y channel

                xCell = cells.at(i);
                yCell = cells.at(j);
                double result = Tim::mutualInformation(range(xCell->begin(),
                                xCell->end()),
                                range(yCell->begin(), yCell->end()),
                                xLag, yLag, K);

                outData.insert(i, j, result);
                outData.insert(j, i, result);
            }
        }
        cells.clear();
    }
```

Listing 5.2: HermesTim mutual information parallelised region

Lines 6 and 7 of listing 5.2 are the same line in real source code because OpenMP directives must be in the same line, it is show in two lines because of readability purposes.

With this parallelisation, the division of the work is done by channels: For channel i we have $n - i$ channels to calculate the mutual information.

Data structures containing one pointer for each trial of each signal are created before the loop region because the structure associated with one channel is used each time a mutual information estimation with this channel is done.

5.1.2 Transfer Entropy

The second wrapper implemented in HermesTim computes the transfer entropy for a signal set. This function uses the transfer entropy estimator defined in the Tim library, this estimator computes the transfer entropy for each pair of signals $T(X, Y)$.

The transfer entropy can be calculated by

$$T(w, X, Y) = H(w, X) + H(X, Y) - H(X) - H(w, X, Y) \tag{5.2}$$

where H is the Shannon differential entropy and w is the future of signal X.

The header definition of the HermesTim transfer entropy function is displayed in listing 5.3. The input parameters defined are: `data` is a signals dataset memory pointer, as defined in 4; `dt` is the time delay, that is the number of samples that signal X will be displaced to get the future of signal X; `samples`, `channels` and `trials` that are the numbers of samples, channels and trials, respectively, of `data`; Finally, `xLag`, `yLag` and `K` are function dependant parameters, as defined in Tim's transfer entropy estimator function.

```
    void transfer_entropy(double *data, int dt, double *results,
                          int samples, int channels, int trials,
                          int xLag, int yLag, int K);
}
```

Listing 5.3: HermesTim transfer_entropy declaration.

As well as with mutual information, the parallelisation is done using OpenMP, in particular using the *parallel for* clause. The work unit shared out between OpenMP threads correspond with the estimation of transfer entropy for two signals. The implementation of transfer entropy is similar to mutual information implementation, with the differences being localised in the estimator used and in the preparation of the w data. Unlike in mutual information, in transfer entropy

$$T(X, Y) \neq T(Y, X) \tag{5.3}$$

because of this the number of estimations needed by a signal set with n signals is n^2.

The source code for the parallelised block is shown in listing 5.4. In the same way as mutual information the auxiliar data structures with pointers for each trial in a signal are created before the parallel region. This allows to reuse the signal pointers every time a signal is used to compute the transfer entropy, increasing performance.

```
.....
       /* Compute Transfer Entropy */
       for(int i = 0; i < channels; ++i)
#pragma    omp parallel for default(none) shared(i, cells, wCells, xLag, yLag,
                    K) schedule(runtime)
           for(int j = 0; j < channels; ++j){
               std::vector<Tim::SignalPtr> *xCell; // Cell for x channel
               std::vector<Tim::SignalPtr> *yCell; // Cell for y channel
               std::vector<Tim::SignalPtr> *wCell; // Cell for w of x channel

               xCell = cells.at(i);
               yCell = cells.at(j);
               wCell = wCells.at(i);
```

```
                double result = Tim::transferEntropy(
16                          range(xCell->begin(), xCell->end()),
                            range(yCell->begin(), yCell->end()),
                            range(wCell->begin(), wCell->end()),
                            xLag, yLag, K);
                resultMatrix.insert(j, i, result);
21          }
.........
```

Listing 5.4: HermesTim transfer entropy parallelised region.

Unlike HermesTim's mutual information algorithm (listing 5.2), in the transfer entropy algorithm it is neccesary to have the future of signal X to calculate the transfer entropy of X and Y. They are calculated for each signal at the same time that the auxiliar data structures with trials pointers are created. To get the future of a signal, it is displaced `dt` samples.

5.2 HermesTim: MATLAB Integration

HermesTim library has been created to be mainly used from MATLAB. Because of this, we are centred in the usability from MATLAB. Therefore, we choose to have only one gateway function for all the services included in the library, this gives us the possibility to only have one compile MEX file in each platform for all the services offered.

A new software software layer is added to HermesTim software architecture (figure 5.1) to be able to use it from MATLAB. This layer is made of a C++ and MATLAB components: C++ component includes a C++ file with the gateway function and one adaptor wrapper for each function that adapts MATLAB data types to C data types, and calls the corresponding HermesTim function; The MATLAB component includes definitions and a function for each service offered by HermesTim as a MATLAB function. This new software layer with the HermesTim library is called HermesTim_Matlab, the complete architecture is shown in figure 5.2.

To use all of the services offered by HermesTim with only one entry point, HermesTim_Matlab defines an enumerate that identifies the requested service. This enumerate is the first parameter that the gateway function receives, and depending on the value a function is chosen, passing the remaining parameters to the adaptor function. This enumerate is both defined in the C++ side and the MATLAB side, and must have the same value for the same identifier, ensuring the right service (see listings 5.5 and 5.6).

```
classdef hermestim
    % This class define the int constant associate with each method given
3   % by hermestim library.

    properties (Constant)
        mutual_information = 0;
        transfer_entropy = 1;
8   end
```

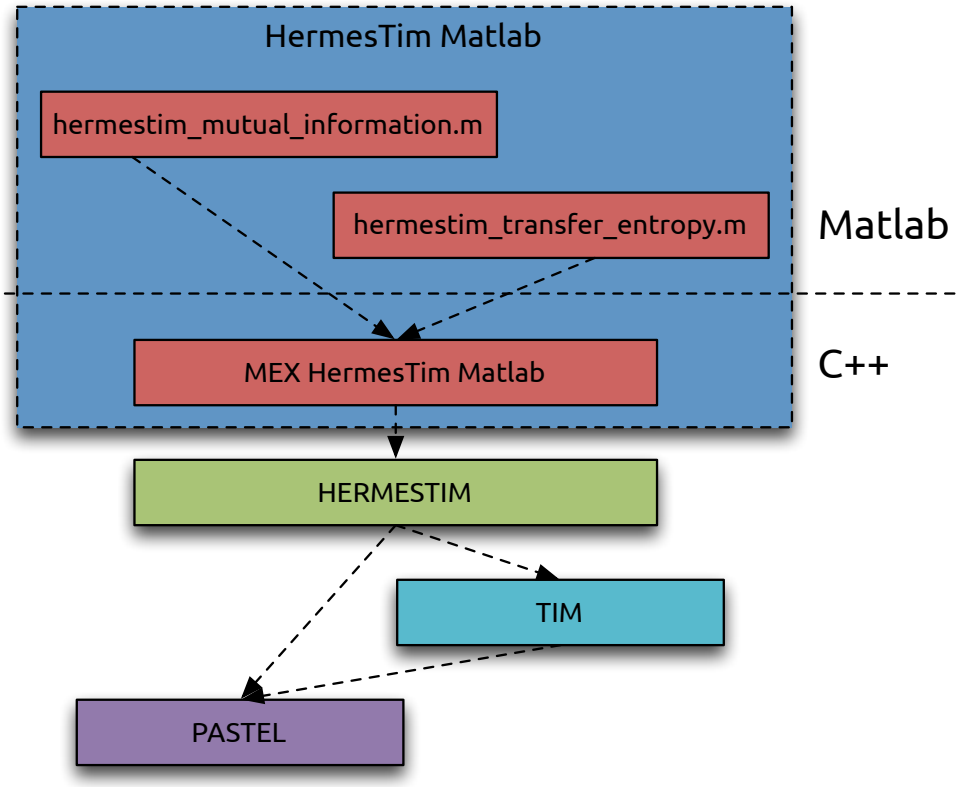

Figure 5.2: HermesTim_Matlab software architecture.

```
end
```

Listing 5.5: Services offered in MATLAB side. `hermestim.m`

```
1 namespace HERMESTim_Matlab{ enum{MUTUAL_INFORMATION, TRANSFER_ENTROPY};
    ...
  }
```

Listing 5.6: Services offered in C++ side. `hermestim_matlab.h`

Hitherto only two services are offered by HermesTim_Matlab, but this architecture allows to add new services to the library easily.

The C++ part is made of the gateway function, `mexFunction`(section 3.3.1), written in C++ and the methods used to adapt data types from MATLAB to C++ for each offered service. The gateway function is responsible of checking the required service, based on the identifier given as the first parameter, and call the adaptor method of the corresponding function. Adaptor methods are responsible for adapting the types of parameters given as arguments, that are MATLAB types, to the types required by the corresponding HermesTim function, and also is responsible of requesting dynamic memory and type checking. The HermesTim mutual information adaptor method is displayed in listing 5.7.

```
namespace HERMESTim_Matlab {
    void mutual_information_mex(int nlhs, mxArray *plhs[],
                          int nrhs, const mxArray *prhs[]){
        double *data, *outData;
        mwSize nChannels, nSamples, nTrials;
        mwSize nDim;
        const mwSize *dimensions;
        int xLag, yLag, K;

        try{
            /* Check the correct number of inputs */
            if(nrhs != 4)
                mexErrMsgTxt("Four input arguments required\n"
                             "Data, xLag, yLag, K");

            /* create a C pointer of the input matrix */
            data = mxGetPr(prhs[0]);

            /* get number of dimensions of the input matrix */
            nDim = mxGetNumberOfDimensions(prhs[0]);
            if (nDim != 3 && nDim != 2)
                mexErrMsgTxt("Array with a different number
of dimension of 2 or 3");

            /* get the array with the dimensions */
            dimensions = mxGetDimensions(prhs[0]);
            nSamples = dimensions[0];
            nChannels = dimensions[1];

            if (nDim == 3)
                nTrials = dimensions[2];
            else
                nTrials = 1;

            /* Get xLag and yLag parameters */
            xLag = (int) mxGetScalar(prhs[1]);
            yLag = (int) mxGetScalar(prhs[2]);

            /* Get number of neighbours */
            K = (int) mxGetScalar(prhs[3]);

            /* set the output pointer to the output matrix */
            plhs[0] = mxCreateDoubleMatrix(nChannels, nChannels,
                                           mxREAL);

            /* create a C pointer to output matrix */
            outData = mxGetPr(plhs[0]);

            ......
            HERMESTim::mutual_information(data, outData,
                                          nSamples, nChannels, nTrials,
                                          xLag, yLag, K);
```

.....

Listing 5.7: HermesTim_Matlab mutual information adaptor method.

The adaptor method implementation is simple: it first checks the number and type of the input parameters; then tries to obtain the C++ equivalent types of MATLAB types, with simple types the process is easy using the MATLAB API, but with complex data types this has more difficulty, this was explained at the beginning of chapter 4. It is necessary to know how the values are stored in memory to use this directly, if not, it is neccesary to do copies and conversions of the data. In our case it is only necessary to obtain the pointer to the first element of the dataset. All the development has been made considering how MATLAB manages the memory, no conversions or copies of MATLAB Matrix are needed.

In the MATLAB part of HermesTim_Matlab the identifiers and a MATLAB level wrapper of the services offered to the user are defined. These wrappers are MATLAB functions that basically implement type checking and call to MEX file with the corresponding identifier and parameters. In the case of mutual information service, a MATLAB M function named `hermestim_mutual_information` has been created within the *hermestim_mutual_information.m* file, this way a user only needs to call to *hermestim_mutual_information* function with the correct parameters to get the results, making sure first that HermesTim_Matlab MEX compiled file is in MATLAB path. For transfer entropy, the same process has been followed.

Given a MATLAB M function for each service gives a comprehensible meaning to users, and allows to include more functionality easily in each service on the MATLAB side. The implementation of mutual information in M language is showed in listing 5.8

```
% HERMESTIM_MUTUAL_INFORMATION
% Data is the data to calculate mutual information.

% .....

% Optional input arguments in 'key'-value pairs:
%
% XLAG and YLAG ('xLag', 'yLag') are integers which denote the amount of
% lag to apply to signals. Default 0.
%
% K ('k') is an integer which denotes the number of nearest neighbours to
% be used by the estimator. Default 1.

function [ MI ] = hermestim_mutual_information( data, varargin)
% Input parser
p = inputParser;

% Optional input arguments
defaultXLag = 0;
defalutYLag = 0;
defaultK = 1;
```

```
% Prepare parse arguments
addRequired(p, 'data');
addParamValue(p, 'k', defaultK,@isnumeric);
addParamValue(p, 'xLag', defaultXLag, @isnumeric);
addParamValue(p, 'yLag', defalutYLag, @isnumeric);

% Parse input arguments
parse(p, data, varargin{:});


MI = hermestim_matlab(hermestim.mutual_information, p.Results.data, ...
                    p.Results.xLag, p.Results.yLag, p.Results.k);

end
```

Listing 5.8: MATLAB implementation of hermestim_mutual_information.

The MATLAB implementation defines default values for the different parameters allowed and checks types for input arguments, it then calls *hermestim_matlab* MEX library with the identifier to calculate the mutual information as first argument and the remaining arguments. The transfer entropy implementation is similar, the difference is that transfer entropy declares one more parameter, `dt` that is the displacement of the future of signal X.

5.2.1 HermesTim_Matlab compilation

We have created a set of MATLAB scripts to help the user compile the HermesTim_Matlab MEX library. Three versions are included depending on the parallelism level: Sequential, OpenMP parallelisation only in HermesTim library and OpenMP in HermesTim, Tim and Pastel library. These scripts are user machine dependant as the path to necessary libraries and headers usually changes between different machines. To compile the library it is necessary to modify the predefined paths within the script, these paths are the TimCore, PastelSys and HermesTim libraries and HermesTim and HermesTim_Matlab headers files. When the paths are modified the user only needs to execute the script in a MATLAB console to build HermesTim_Matlab MEX library.

A diagram with the compilation process is shown in figure 5.3.

5.3 HermesTim: Parallelisation Results

To get results of the new library, four computers with different architectures have been used. One of the computers has an OS X operating system and the rest use a 64-bit Linux. Technical characteristics of the computers used in the different tests and a tag name associated are listed in the table 5.1.

In all the tests, the same compilation flags are used to compile the libraries: Pastel, Tim and HermesTim. These flags are:

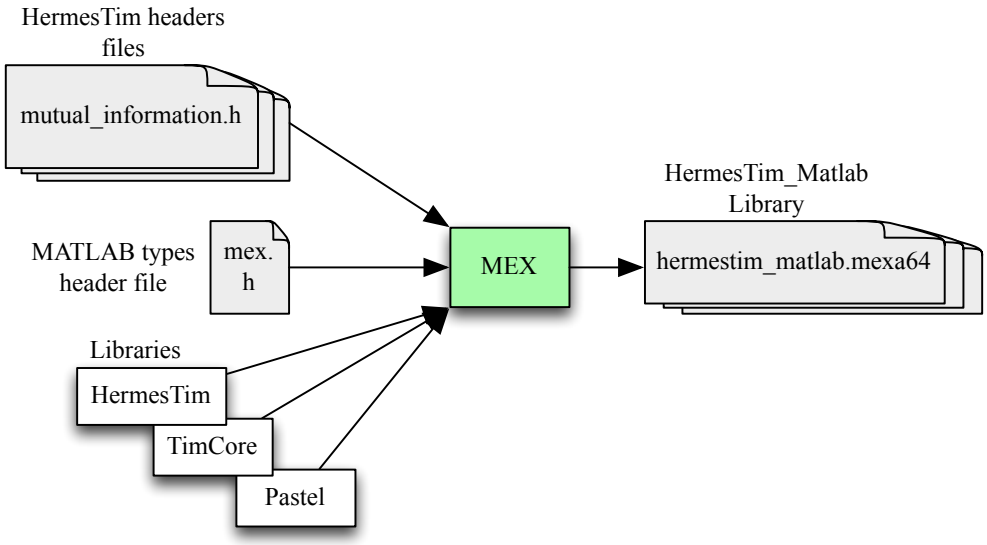

Figure 5.3: HermesTim_Matlab compilation process.

Name	Processors	N Processor	N Cores	RAM Memory	GCC Version
Ebano	AMD FX-8350	1	8	16GB	4.6
Espino	Xeon 5500	2	12	48GB	4.4
Magnolio	Opteron	4	48	16BG	4.4
MacBook	Core2Duo P8600	1	2	8GB	4.7

Table 5.1: Test computers.

- -O2: Optimisation level 2, according to GCC Manual[8], the compiler performs nearly all supported optimisation's that do not involve a space-speed tradeoff.
- -g: Produce debugging information.

The measures are ne over three different scenarios, each with a different size dataset:

- Scenario 1: Two seconds of real MEG with a frequency of 1000Hz. 100 channels and 2000 samples per channel.
- Scenario 2: Five seconds of real MEG with a frequency of 1000Hz. 150 channels and 5000 samples per channel.
- Scenario 3: Two seconds of real MEG with a frequency of 1000Hz, analysis repeated 30 times. 100 channels and 2000 samples per channel, with a total of 30 *trials*.

The scenarios and computers used are the same used in the previous analysis (section 4). These tests consist of obtain the execution time used to estimate the mutual information for each channel pair in the dataset.

We use the speedup to measure the improvement reached. Speedup refers to how much parallel algorithm is faster than a corresponding sequential algorithm. And is defined by the equation 5.4.

$$S_p = \frac{T_1}{T_p} \tag{5.4}$$

Where:

- p is the number of processors.
- T_1 is the execution time of the sequential version.
- T_p is the execution time of the parallel version.

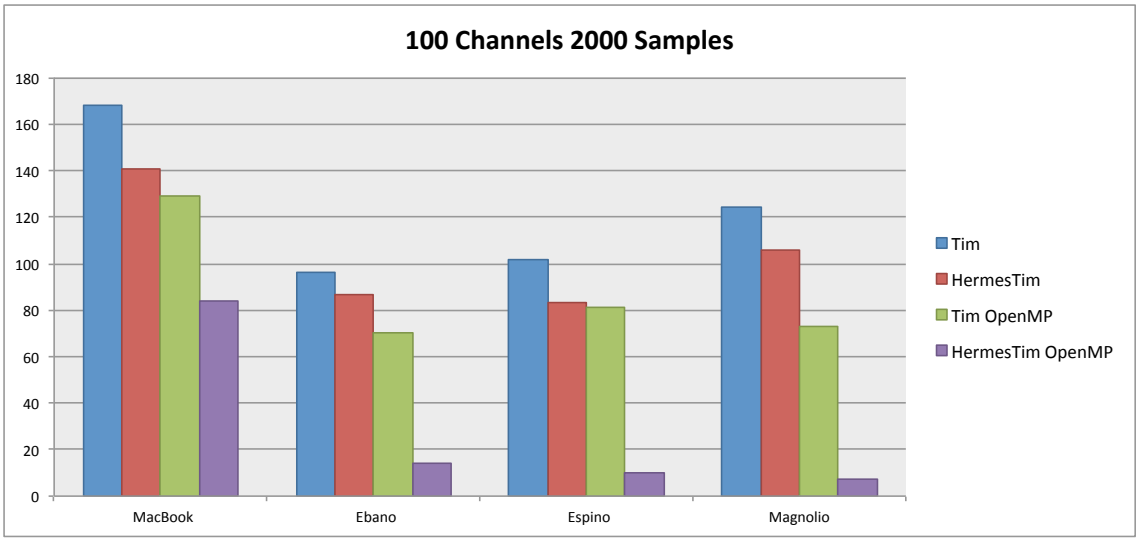

Figure 5.4: MI execution times from MATLAB: Scenario 1.

Figures 5.4, 5.5 and 5.6 show histograms with the absolute number of seconds used to compute mutual information for each pair of channels in the dataset.

The time is measured for four different versions: the sequential and parallel version of TIM; and the sequential and parallel version of HermesTim.

The two sequential versions are called *Tim (seq)* and *HermesTim (Seq)*. *Tim (Seq)* version matches the original version where the channel selection loop is implemented in MATLAB, and the mutual information function of Tim library, developed in C++, is called within this loop region for each pair of channels. *HermesTim (Seq)* version matches the new library developed, called HermesTim, with this library all the work out is done in C++.

In the same way as sequential versions, we have two parallel versions of the libraries: *Tim (OpenMP)* that is equivalent to *Tim (Seq))* version, where the selection of each pair of channel is implemented in MATLAB and Tim mutual information estimator is called, but in this case the libraries Tim and Pastel are compiling with parallelisation switched on; and *HermesTim (OpenMP)* version where HermesTim

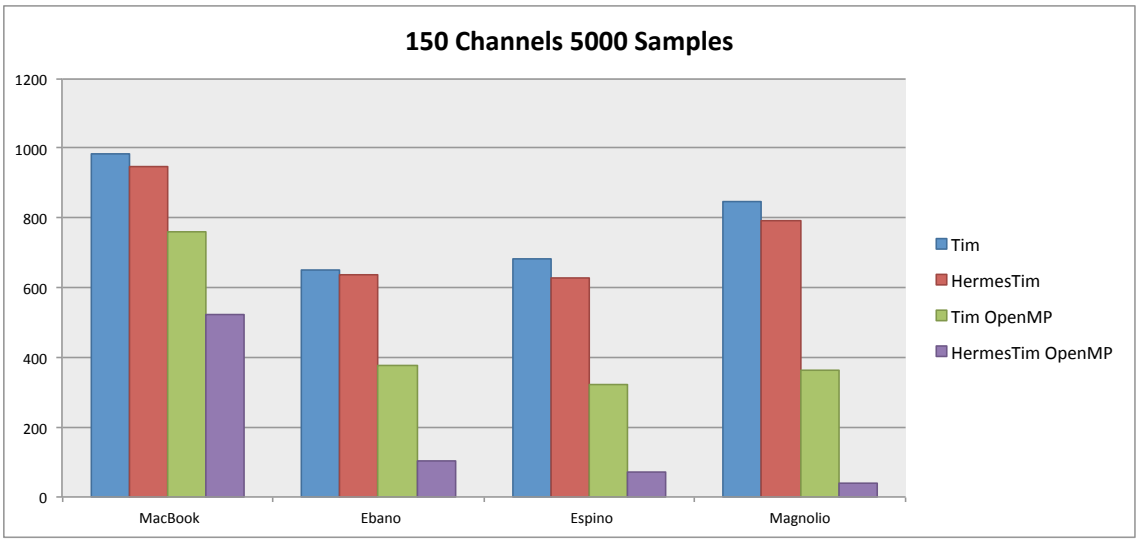

Figure 5.5: MI execution times from MATLAB: Scenario 2.

library is compiled with parallelisation activated but Pastel and Tim libraries are compiled without parallelisation.

Figures 5.5 and 5.6 show the times in seconds for scenarios 2 and 3 respectively. The scenario 3 has not been measured in MacBook computer because is the slower machine and the dataset used is the biggest, needing a lot of time to get the results. Furthermore, the results would not help with the conclusions because the ideal speedup of MacBook is 2, because the the low number of processors.

Measured times and the corresponding speedup for each test are displayed in table 5.2. The speedup is calculated using the corresponding sequential version for each parallel version, which means that for parallel time obtained from *Tim (OpenMp)* parallel version the equivalent sequential time is the obtained from *Tim (Seq)* version in the same scenario, and for *HermesTim (OpenMP)* the sequential version *HermesTim (Seq)* is used.

Comparing the sequential version it is observed that the necessary time to calculate the mutual information with the new developed library is lower than the original implementation. This difference of time (showed in charts in figures 5.4, 5.5 and 5.6 or from table 5.2) is the profit achieved using the C++ implementation instead of the MATLAB implementation for the channel pair selection loop region, the increase of performance is between 10 and 300 seconds.

The obtained speedup for the *Tim (OpenMP)* version is around 2 for scenarios without trials, the speedup obtained in the scenarios with trials is greater but far from ideal speedup, $5,85$ in Espino, $8,92$ in Magnolio and $3,69$ in Ebano. The ideal speedup for MacBook, Ebano, Espino and Magnolio are 2, 8, 12 and 48 respectively.

HermesTim gives us better speedup in the scenarios without trials, around 70% of improvement: $1,68$ and $1,80$ in MacBook; $6,86$ and $6,25$ in Ebano; $8,30$ and $8,85$ in Espino, and around 50% of improvement in Magnolio: $15,14$ and $19,29$. But in the scenario where datasets have trials, scenario 3, the new library has a

	Scenario	Version	Sequential T.	Parallel T.	SpeedUp
MacBook	Scenario 1	Tim	168	129	1.30
		HermesTim	141	84	1.68
	Scenario 2	Tim	985	762	1.29
		HermesTim	945	525	1.80
Ebano	Scenario 1	Tim	96	70	1.37
		HermesTim	87	14	6.21
	Scenario 2	Tim	653	378	1.73
		HermesTim	638	102	6.25
	Scenario 3	Tim	11700	3174	3.69
		HermesTim	11380	4320	2.63
Espino	Scenario 1	Tim	102	81	1.26
		HermesTim	83	10	8.30
	Scenario 2	Tim	682	323	2.11
		HermesTim	628	71	8.85
	Scenario 3	Tim	12840	2195	5.85
		HermesTim	12660	2580	4.91
Magnolio	Scenario 1	Tim	124	73	1.70
		HermesTim	106	7	15.14
	Scenario 2	Tim	848	365	2.32
		HermesTim	791	41	19.29
	Scenario 3	Tim	21480	2409	8.92
		HermesTim	21120	1931	10.94

Table 5.2: MI: Speedup and times in seconds for each scenario.

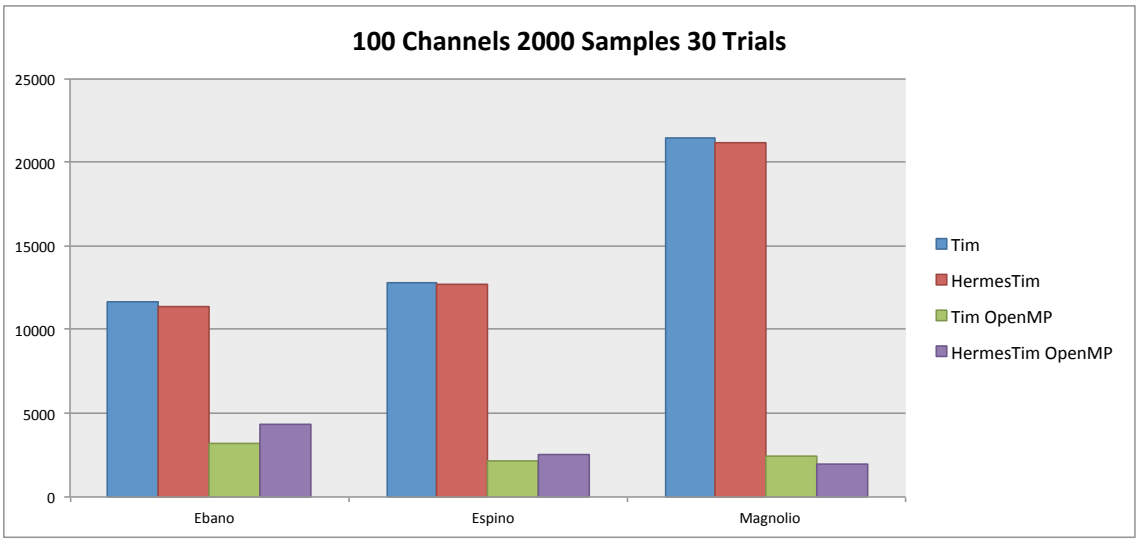

Figure 5.6: MI execution times from MATLAB: Scenario 3.

slight performance loss compared with *Tim (OpenMP)*. Although, the latter gives lower values.

After the results have been analysed, we think that this parallelisation causes memory management problems due to memory bottlenecks and the OpenMP default scheduling policy used. The worst time with the new library should not be worse than with the original library.

5.4 OpenMP scheduling analysis

The obtained results using HermesTim library from HERMES, presented in section 5.3, give us the idea that coarse-grain parallelisation done in HermesTim is not working as well as we were expecting.

We do two further analyses to have a better idea why the parallelisation is not working well: An OpenMP schedule policy analysis for HermesTim and a maximum speedup analysis for each machine used in the tests.

5.4.1 OpenMP Schedule analysis

OpenMP standard[20] defines three principal scheduling policies:

- `guided`: The iterations are assigned to threads in the team in chunks as the executing thread request them. The size of each chunk is proportional to the number of unassigned iterations divided by the number of threads in the team, decreasing to 1.

- `dynamic`: The iterations are distributed to threads in the team in chunks as the threads request them. Each thread executes a chunk of iterations, then request another chunk, until no chunks remain to be distributed.

- `static`: The iterations are divided into chunks of specified size, and the chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number. If no chunk size is specified, the iteration space is divided into chunks that are approximately equal size, and at most one chunk is distributed to each thread.

To make the analysis we use other scheduling policy named `runtime`, that enforces to use the scheduling policy specified in a environment variable, called `OMP_SCHEDULE`, or a default policy if this environment variable is not declared. This helps to test the different policies without changing the source code. Furthermore, to make the analysis, we use a C++ program that we have created for this purpose, that uses HermesTim library and reads the data from a MATLAB matrix file. This program imitates the behaviour of MEX mutual information of HermesTim_Matlab, that makes types conversions and calls mutual information function from HermesTim.

The size of the data used is 40 channels with 1000 samples and 30 trials. This gives enough work to be assigned to parallel region and declines the required time to get the results.

Two different work grains are used for this test:

- *Channel pair*: Each iteration in a parallel region computes the mutual information for each channel pair. But the OpenMP scheduler is executed for each channel, the algorithm selects the i channel and in the parallel region each iteration calculates the mutual information of i and j where $j > i$. Listing 5.9.

- *Channel*: Each iteration in the parallel region calculates the mutual information of i and a set of channels. Each set is formed by j where $j > i$. Listing 5.10

```
.........
       for(int i = 0; i < channels; ++i)
       {
           outData.insert(i,i,1); // MI of the same channel

           #pragma omp parallel for default(none) shared(cells, outData,
               channels, xLag, yLag, K, i) schedule(runtime)
           for (int j = i + 1; j < channels; ++j){
               /* Calculate mutual information for pair of channels */
               ..........
           }
       }
.........
```

Listing 5.9: *Channel pair* workload parallelisation.

```
#pragma omp parallel for default(none) shared(cells, outData,
    channels, xLag, yLag, K) schedule(runtime)
      for(int i = 0; i < channels; ++i)
```

```
        {
            outData.insert(i,i,1); // MI of the same channel

            for (int j = i + 1; j < channels; ++j){
                /* Calculate mutual information for pair of channels */
                .........
            }
        }
.........
```

Listing 5.10: *Channel* workload parallelisation.

The results for *channel pair* are shown in table 5.3, and for *channel* are shown in table 5.4. These are the sequential time, and parallel time for each scheduling policy and the speedup obtained.

		Sequential	`static`		`dynamic`		`guided`	
Ebano	Real	514.39	142.172		135.792		138.068	
	User	511.00	919.237	3.62	929.198	3.79	929.778	3.73
	Sys	2.80	4.092		4.328		4.272	
Magnolio	Real	935.04	76.249		98.99		98.624	
	User	932.56	1665.84	12.26	1996.61	9.45	1976.17	9.48
	Sys	2.30	46.03		41.75		38.98	
Espino	Real	672.34	115.16		120.052		113.475	
	User	669.00	1869.89	5.84	1942.23	5.60	1825.46	5.93
	Sys	2.10	19.67		19.6		4.23	

Table 5.3: *Channel pair*: times in seconds and speedup.

		Sequential	`static`		`dynamic`		`guided`	
Ebano	Real	514.39	138.716		120.602		189.385	
	User	511.00	793.838	3.71	824.288	4.27	717.025	2.72
	Sys	2.80	3.944		3.936		3.904	
Magnolio	Real	935.04	75.459		82.172		81.26	
	User	932.56	1719.16	12.39	1851.83	11.38	1775.32	11.51
	Sys	2.30	30.25		28.28		22.99	
Espino	Real	672.34	140.413		83.563		142.995	
	User	669.00	725.6	4.79	902.58	8.05	799.04	4.70
	Sys	2.10	2.13		1.26		2.03	

Table 5.4: *Channel*: times in seconds and speedup.

OpenMP schedule is executed c times in the *channel pair* test, and in each execution the quantity of iterations change, the first time there are 40 iterations, that match with the number of channels, and in each execution the number of

iterations decrease in one unit. OpenMP parallel regions have an internal wait barrier so that all threads wait for all of the work is done in the parallel region, this has the inconvinience of free threads having to wait other threads to finish their work. This is not enough work for Magnolio, but is sufficient for the other computers.

In *channel* test, the OpenMP scheduler is executed only one time, with a total of 40 iterations, each iteration has different volume of work. In this case, it makes sense that the worst scheduling policy was `static` in all the computers, but this is not the case with Magnolio, this is possibly due to the lack of work. In the other computers the best scheduling policy for this test is `dynamic`, because when a thread finished its work requested another iteration, as the iterations have different work volumes the final iterations will need less time and will be scheduled to free threads while the bigger volume work iterations are being calculated.

The computers used for the analysis are shared with other users and this is the cause that the measured times are not always the same, but are normally very similar. Furthermore, we realise the same test setting the maximum number of threads that OpenMP can use in each parallel region to the number of processor minus one, leaves one processor for operative system tasks, this gives slightly better results in Espino and Ebano but not in Magnolio, where it has a lot of processors.

As conclusion the best scheduling policy is `dynamic` with a chunk of 1. But we find that the distribution of work is not the best in any of the cases analysed. To solve this problem we propose a different distribution of work, mixing the two solutions studied here. This solution should execute the OpenMP scheduler only one time with the computation of mutual information for each channel pair as work unit (see section 5.5.1).

5.4.2 Maximum SpeedUp Analysis

Although the ideal speedup of each machine is known, this speedup can be affected by many different things like: memory bottlenecks, cache problems, CPU architecture, etc. Because of this, we decide to do a maximum speedup analysis to the different computers, executing a very high parallelised program that uses data that can hold in cache memory during the execution, avoiding memory bottlenecks and cache block replacements.

We decided to make a matrix multiply program, because it is an inherent parallelisation algorithm, and as iteration work unit we chose a row of a result matrix. To measure the biggest speedup it is necessary that all the data needed by a thread to do the work fits in cache memory avoiding cache replacements.

The work grain calculates a whole row of the result matrix, being the data used the row m of matrix A, the whole matrix B and the row m of the result matrix.

The source code of the matrix multiplication algorithm in C++ is presented in listing 5.11. The OpenMP parallel directive is set in the first loop block defining a result row as work unit for the OpenMP thread.

. . . . . .

```
#pragma omp parallel for default(none) shared(mat1, mat2, matR) schedule(runtime)
        for (int i=0; i<M1; i++)
                for (int j=0; j<N2; j++) {
                        int acum = 0;
                        for (int k=0; k<N1; k++) {
                                acum += mat1[i][k] * mat2[k][j];
                        }
                        matR[i][j] = acum;
                }
        }
```

Listing 5.11: C++ parallel matrix multiplication implementation

The test has been done twice with different sizes of matrix:

- Test1: M1($4096x1536$) with a size of 48MB, row has 12KB. M2($1536x512$) with a size of 6MB. Size of the result matrix row is 4KB.
- Test2: M1($8192x512$) with a size of 32MB, row size is 4KB. M2($512x1024$) with a size of 4MB. Size of the result matrix row is 8KB.

	Sequential	Parallel	SpeedUp
MacBook	91.953	68.302	1.35
Ebano	34.138	5.292	6.45
Espino	29.035	2.71	10.71
Magnolio	74.975	10.076	7.44

Table 5.5: Maximum speedup analysis: test 1.

	Sequential	Parallel	SpeedUp
MacBook	28.301	15.389	1.84
Ebano	33.382	5.719	5.84
Espino	37.361	3.447	10.84
Magnolio	77.719	1.928	40.31

Table 5.6: Maximum speedup analysis: test 2.

The maximum SpeedUp of Ebano and Espino are very similar in the two tests (tables 5.5 and 5.6), this is because they have enough cache memory to keep the data in both tests, 8MB in Ebano and 12MB in Espino. MacBook only has 3MB of L2 cache and when the M2 matrix is bigger the SpeedUp is lower. The oddest is Magnolio, that is made up of 4 Opteron 6176SE, these processors have 2x6MB of L3 cache and seeing the results the cache is not managed in right way, in the case data size is less than 6MB the SpeedUp is ~ 40, and when the test data is bigger than 6MB the SpeedUp is only ~ 7, we can see that Magnolio is penalised when data is not in cache memory.

	MacBook	Ebano	Espino	Magnolio
SpeedUp	1.84	6.45	10.84	40.31

Table 5.7: Maximum speedup achieve in each computer.

The maximum speedup measured for each machine is shown in table 5.7. This real maximum speedup gives us an element to compare results. Comparing with the speedup showed in the parallelisation results (table 5.2), the maximum speedup in scenarios with trials are far from the maximum speedup gotten in this test for all the machines, however in scenarios without trials the maximum speedup of Ebano, Espino and MacBook come to the maximum speedup measured in this tests.

5.5 HermesTim: Improvements

With the obtained results from OpenMP analysis (section 5.4) it has been concluded that the HermesTim library can be improved. The improvements suggested and implemented cover the OpenMP scheduler and take advantage of the internal parallelisation of the Tim library. Regarding the OpenMP planning, we propose a scheduling plan where all the computation for each channel pair is the work to share out, executing the scheduler only once (section 5.5.1). The other improvement we propose is to exploit the internal parallelisation of the Tim library joining it with the parallelisation implemented in HermesTim, this is called nested parallel regions in OpenMP (section 5.5.2).

5.5.1 Collapse clause

OpenMP standard defines the *collapse* clause. This clause may be used to specify how many loops are associated with the OpenMP loop construct, *omp parallel for*. If more than one loop is associated with the loop construct, then the iterations of all associated loops are collapsed into one larger iteration space.

The *collapse* clause does what we are looking for, but this clause has limits. One of the limits that affect HermesTim current implementation is that the iteration count for each associated loop is computed before the entry to the outermost loop. This limit collides with the mutual information algorithm implementation written in HermesTim, where the iteration count of the second loop depends on the first loop counter, but not in the transfer entropy implementation. To avoid this problem a conditional clause if added within the innermost loop region, executing the sentences only if the mutual information for the channel pair has not been computed yet.

Other problem of the use of this clause is that work sentences only can be in the innermost loop, but in HermesTim the outermost loop clause initialises the matrix diagonal with the mutual information value for each channel with itself that are 1. Now this initialisation is done after the parallel region. The new solution for the parallel region is shown in listing 5.12.

```
.........
#pragma omp parallel for collapse(2) default(none) shared(cells, outData,\
channels, xLag, yLag, K) schedule(runtime)
for(int i = 0; i < channels; ++i)
          for (int j = i + 1; j < channels; ++j){
#ifdef _OPENMP
              if (j > i){
#endif
              std::vector<Tim::SignalPtr> *xCell; // Cell for x channel
              std::vector<Tim::SignalPtr> *yCell; // Cell for y channel
              /* Create a signal for each trial of signal x and y and add it
                 to the correspond cell
              */
              xCell = cells.at(i);
              yCell = cells.at(j);
              double result = Tim::mutualInformation(range(xCell->begin(), xCell->end()),
                              range(yCell->begin(), yCell->end()),
                              xLag, yLag, K);

              outData.insert(i, j, result);
              outData.insert(j, i, result);
#ifdef _OPENMP
              }
#endif
          }
.........
```

Listing 5.12: HermesTim Mutual information collapse parallel region

The conditional sentence only is taked into account when OpenMP is used.

		Sequential	OpenMP	SpeedUp
MacBook	Real	749.055	758.821	0.99
	User	731.643	1402.083	
	Sys	1.581	6.549	
Ebano	Real	514.39	137.054	3.75
	User	511.00	961.584	
	Sys	2.80	4.868	
Espino	Real	672.34	112.192	5.99
	User	669.00	2660.34	
	Sys	2.10	4.01	
Magnolio	Real	935.04	66.295	14.10
	User	932.56	3026.31	
	Sys	2.30	23.27	

Table 5.8: Times in seconds and speedup using collapse clause.

Table 5.8 displays the times necessary to compute the mutual information for a dataset of 40 channels, 2000 samples and 30 trials, the same used in OpenMP

scheduling policies analysis (section 5.4), but in this case only the `dynamic` policy has been used. If we compare this results with *channel pair* parallelisation test results (table 5.3), it is observed that the speedup measured for Ebano and Espino are very similar, but in the case of Magnolio the speedup goes up from 9,45 to 14,10. But if we compare with results from *channel* parallelisation test (table 5.3), Magnolio obtained better SpeedUp (+24%), Ebano obtained slightly low SpeedUp (−14%), and Espino has a lower SpeedUp (−34%).

The collapse clause gives a better performance in Magnolio, and does not change the performance in the other machines. But it is possible to get better results becasuse these are far from the results obtained with HermesTim in scenarios without trials.

5.5.2 Nested Parallelised Regions

OpenMP library only parallelise the first parallel region by default, that means that if a parallel region exists within the parallelised region then the last is not parallelised by default. But this behaviour can be changed configuring OpenMP library using the OpenMP API or the environment variable `OMP_NESTED`.

Taking advantage of the nested parallel feature given by OpenMP it is possible to improve the results of HermesTim. This characteristic can be used to use the parallelised region of HermesTim and Tim libraries at the same time. The possible improvement is not known, but it can be measured using a version of HermesTim that links with the parallelised version of TIM. The OpenMP `omp_set_nested` function is used in order to ensure that nested parallel regions are activated, this function is included within a conditional define that only works when OpenMP is used to compile, this allows to compile the same source code as sequential or parallel, see listing 5.13.

```
.......
#ifdef _OPENMP
       omp_set_nested(1); // Enable OpenMP nested regions
#endif
.......
```

Listing 5.13: Nested parallel regions activation.

The nested parallel version of HermesTim is used with the same dataset used in the previous tests, made up of 40 channels, 1000 samples and 30 trials, to measure the improvement. In table 5.9 the results of mutual information computed for the dataset with the version with collapse clause with nested parallel regions and without are compared.

The results obtained with nested parallel regions are good in all the machines. Magnolio had the same value 14, but this seem to be produced by a memory bottleneck. Espino got a speedup of 10,36 that is close to the ideal speedup, 12, and closest to the maximum speedup measured in the maximum speedup analysis done. Ebano obtained 4,46 of SpeedUp that is not a great value, but it is not far from the maximum speedup measured.

		Sequential	OpenMP	SpeedUp	FullOpenMP	SpeedUp
	Real	749.055	758.821		578.896	
MacBook	User	731.643	1402.083	0.99	1095.74	1.29
	Sys	1.581	6.549		7.963	
	Real	514.39	137.054		115.282	
Ebano	User	511.00	961.584	3.75	832.712	4.46
	Sys	2.80	4.868		2.048	
	Real	672.34	112.192		64.886	
Espino	User	669.00	2660.34	5.99	1479.67	10.36
	Sys	2.10	4.01		17.86	
	Real	935.04	66.295		65.623	
Magnolio	User	932.56	3026.31	14.10	2159.82	14.25
	Sys	2.30	23.27		157.02	

Table 5.9: Times in seconds and speedup using nested parallel regions.

This new improvement improved the Hermestim library performance in all the computers, but the volume of improvement is not the same in all of them. The architectures of the different computers are very different, mono-processor versus multi-processors, cache memory hierarchy, etc. These have a big influence in the results.

Other feature specified by OpenMP standard that could improved the results is the use of binding processor. With this option activated the execution environment should not move OpenMP threads between processors, making the cache data more coherent, avoiding a lot of cache misses. But it has not been possible to test this option, because it is introduced in OpenMP 3.1 and the first version of GCC that implement this OpenMP specification is GCC 4.7. The only computer used for the tests wich GCC 4.7 or greater installed is MacBook and the results are not concluding.

Chapter 6

Results

In this chapter the final results are shown. The libraries (Tim, Pastel and Hermes-Tim) are compiled with optimisations, and all the improvements found as a results of the OpenMP analysis (sectio1on 5.4) are included in HermesTim final source code.

Only three machines of a total of four are used, due to the measurements taken in MacBook, that has a dual core processor, do not allow us to deduce conclusive results. The computers used are listed in table 6.1.

Name	Processor	Processor N	Cores N	RAM Memory	GCC Version
Ebano	AMD FX-8350	1	8	16GB	4.6
Espino	Xeon 5500	2	12	48GB	4.4
Magnolio	Opteron 6176	4	48	16BG	4.4

Table 6.1: Test computers.

In all the tests, the same compilation flags are used to compile the libraries. These are:

- -O3: Optimisation level 3.
- -ffast-math. Enable floating point speed optimisations.

The tests consist of measure the time used to compute the MI and TE from HERMES for different scenarios. Then the speedup is calculate and compared. The obtained results are filtered according to the service provided.

The measures are done to the original library used by HERMES (Tim) with and without parallelisation, called *Tim (Seq)* and *Tim (OpenMP)* respectively, and are done to the new library developed as a result of this project, named Hermes-Tim, also compiled with and without parallel support, called *HermesTim (Seq)* and *HermesTim (OpenMP)* respectively.

6.1 Mutual Information

For mutual information, the measures are done over three different scenarios, each with a different data set size:

- Scenario 1: Two seconds of real MEG with a frequency of 1000Hz. 100 channels and 2000 samples per channel.
- Scenario 2: Five seconds of real MEG with a frequency of 1000Hz. 150 channels and 5000 samples per channel.
- Scenario 3: One seconds of real MEG with a frequency of 1000Hz, analysis repeated 30 times. 40 channels and 1000 samples per channel, with a total of 30 *trials*.

These tests consist of measure the time used to estimate the mutual information for each channel pair in a channel set.

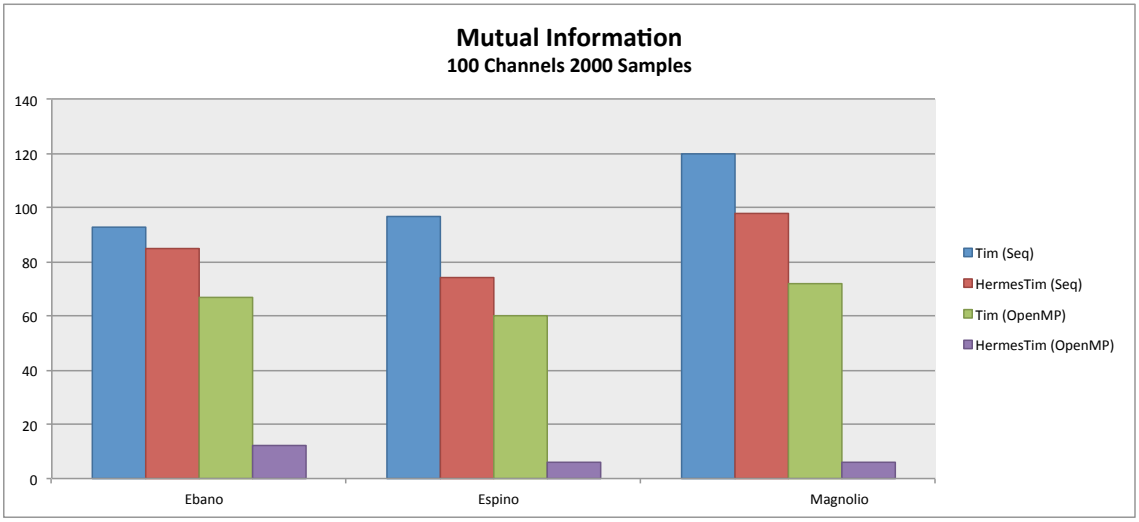

Figure 6.1: MI execution times from MATLAB: Scenario 1.

The figures 6.1, 6.2 and 6.3 show the histograms with the number of seconds used to calculate the mutual information for a whole data set by each version and each computer. Each histogram correspond to a different scenario.

The scenario that contains trials, scenario 3, has a different size that the used in previous analysis section (4) and in parallelisation results section (5.3). This is smaller in order to do more test in the same time period, but the obtained results have the same value.

After the improvements described in this document were included in the HermesTim library, the tests were run and the measurement results were much better than the previous ones. These results are shown in table 6.2. In the case of scenarios without trials, scenario 1 and scenario 2, the speedup achieved is very similar, even a slightly higher. The higher gain is obtained in the scenario with trials.

	Scenario	Version	Sequential T.	Parallel T.	Speedup
Ebano	Scenario 1	Tim	93	67	1.39
		HermesTim	85	12	7.08
	Scenario 2	Tim	642	353	1.82
		HermesTim	637	86	7.41
	Scenario 3	Tim	538	174	3.09
		HermesTim	514	115	4.47
Espino	Scenario 1	Tim	97	60	1.62
		HermesTim	74	6	12.33
	Scenario 2	Tim	648	276	2.35
		HermesTim	556	43	12.93
	Scenario 3	Tim	718	133	5.40
		HermesTim	673	65	10.35
Magnolio	Scenario 1	Tim	120	72	1.67
		HermesTim	98	6	16.33
	Scenario 2	Tim	816	363	2.25
		HermesTim	742	38	19.53
	Scenario 3	Tim	997	136	7.33
		HermesTim	898	58	15.48

Table 6.2: HermesTim Mutual Information results.

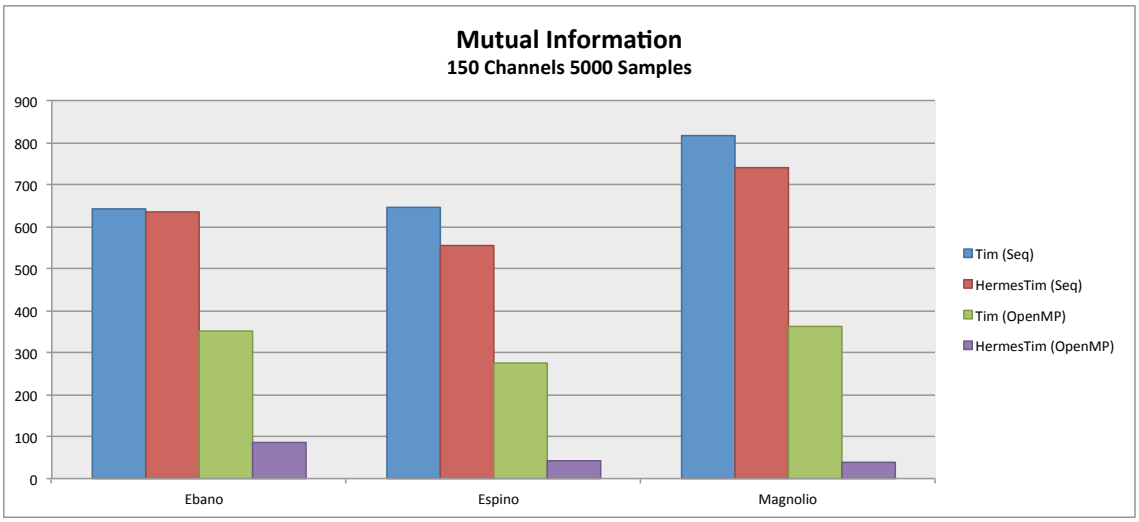

Figure 6.2: MI execution times from MATLAB: Scenario 2.

Comparing the results in the table 5.2, that are the times measured before the improvements are added, and in the table 6.2, it is observed that the speedup in Ebano grew from 2,63 to 4,47, grew from 4,91 to 10,35 in Espino and grew from 10,94 to 15,48 in Magnolio. The improvements give an speedup growth that are between 40% in Magnolio and 110% in Espino.

HermesTim increase the performance of Tim around 5 times in Ebano, $\sim$ 6 times in Espino and $\sim$ 9 in Magnolio, in the scenarios without trials. And $\sim$ 2 times in all the computers in the scenario with trials.

Although the scenario 3 has a smaller dataset size than the dataset used in the preliminary analysis and in the parallel results section, the speedup achieved using the new version of HermesTim with the dataset used in those analysis, that it is composed of 100 channels and 2000 samples per channel and 30 trials are very similar. This was tested in Espino and Magnolio giving a similar performance improvement, but this test was not made with the Tim library due to the big amount of time necessary. Because of this the times and speedup for this dataset are not include in the table with the results.

In summary, the speedup achieved using the new HermesTim library is nearest to the ideal speedup in Espino (12), in Ebano the speedup is far from the ideal speedup in scenarios with trials but it improves the speedup achieved with the original Tim library in the same scenario, and is near in scenarios without trials 7,41 (8). Finally, the speedup achieved in Magnolio, 20 and 15 for datasets with and without trials respectively, has been improved but it continues far from the maximum speedup measured in maximum speedup analysis, that was 40. But it is a good result taking into account the memory problems found in the maximum speedup analysis, where the achieved speedup was 7 when the matrix were big to fit in cache memory.

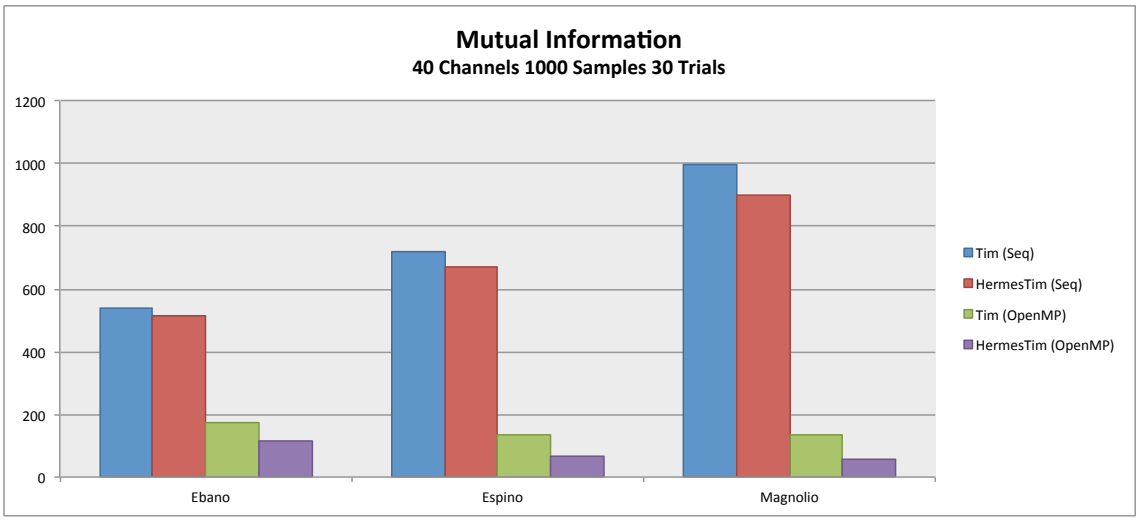

Figure 6.3: MI execution times from MATLAB: Scenario 3.

6.2 Transfer Entropy

The measures are done over three different scenarios, each with a different dataset size:

- Scenario 1: Two seconds of real MEG with a frequency of 1000Hz. 100 channels and 2000 samples per channel.
- Scenario 2: Five seconds of real MEG with a frequency of 1000Hz. 150 channels and 5000 samples per channel.
- Scenario 3: One seconds of real MEG with a frequency of 1000Hz, analysis repeated 15 times. 40 channels and 1000 samples per channel, with a total of 15 *trials*.

These test consist of measured the time used to estimate the transfer entropy for each channel pair in a channel set. This includes the process of calculate the transfer entropy from channel 1 to channel 2, and vice versa, from channel 2 to channel 1.

The figures 6.4, 6.5 and 6.6 show histograms with the measured times, in seconds, used to compute the transfer entropy for a whole dataset. Each figure corresponds to a different scenario.

The two first scenarios are the same that have been used in the mutual information tests, but the third scenario used is slightly smaller. This scenario is selected because the time used to compute the transfer entropy is larger that the time used to compute the mutual information for dataset with the same size, using the same sequential version of the algorithm.

The differences in the histograms between *Tim (OpenMP)* and *HermesTim (OpenMP)* for the transfer entropy are larger than for the mutual information. This is because the speedup obtained for transfer entropy with HermesTim is better in

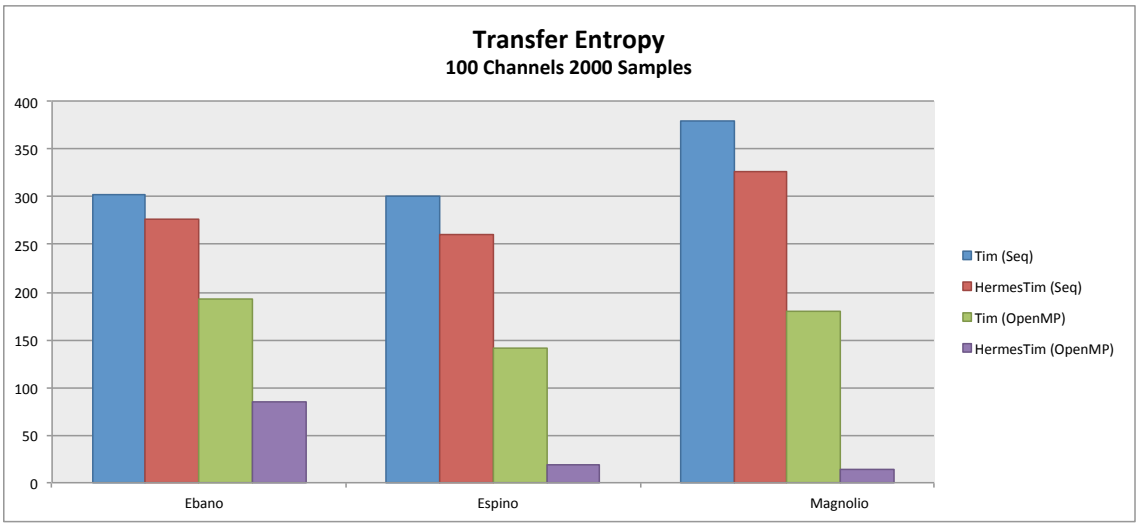

Figure 6.4: TE execution times from MATLAB: Scenario 1.

all the cases. Although the Tim version achieves better speedup for transfer entropy that the achieved for the mutual information, the transfer entropy algorithm of HermesTim achieves up to 10 times speedup improvements that the speedup achieved with Tim.

The measured times and the achieved speedup for the different scenarios is summarised in the table 6.3. The speedup achieved with the new HermesTim library is larger if it is compared with the speedup achieved with the original Tim library. This profit is between 1100% and 500% in Magnolio, 650% and 180% in Espino, and around 200% in Ebano. These are a great values.

Comparing the maximum speedup achieved in the transfer entropy tests, it is near to the maximum speedup measured in maximum speedup analysis. The the achieved speedup in Espino is between $\sim$ 13, in the scenarios that uses dataset without trials, and $10,27$, in the scenario that used dataset with trials, where the ideal speedup for this machine is 12. The speedup achieved in Magnolio is between 23 and 29, in scenarios without trials, and 30, in the scenario that contains trials, these values are near from the maximum speedup measured for this computer (40) in the maximum speedup analysis (section 5.4.2). Finally the achieved speedup in Ebano is between $3,26$ and $4,08$, in the scenarios without trials, and $6,38$, in the scenarios that contains trials, these are close to the maximum speedup measured $(6,45)$.

6.3 Summary

The times and measured speedup for mutual information and transfer entropy are summarised in the tables 6.2 and 6.3 respectively.

It can be observed that The HermesTim library is more scalable in all the cases, because the achieve speedup for the different machines are closest to the

	Scenario	Version	Sequential T.	Parallel T.	Speedup
Ebano	Scenario 1	Tim	302	193	1.56
		HermesTim	277	85	3.26
	Scenario 2	Tim	2238	899	2.49
		HermesTim	2142	525	4.08
	Scenario 3	Tim	786	236	3.33
		HermesTim	759	119	6.38
Espino	Scenario 1	Tim	300	141	2.13
		HermesTim	261	19	13.74
	Scenario 2	Tim	2233	729	3.06
		HermesTim	2056	151	13.62
	Scenario 3	Tim	996	181	5.50
		HermesTim	914	89	10.27
Magnolio	Scenario 1	Tim	380	180	2.11
		HermesTim	326	14	23.29
	Scenario 2	Tim	2845	936	3.04
		HermesTim	2645	89	29.72
	Scenario 3	Tim	1312	217	6.05
		HermesTim	1270	42	30.24

Table 6.3: HermesTim transfer entropy results.

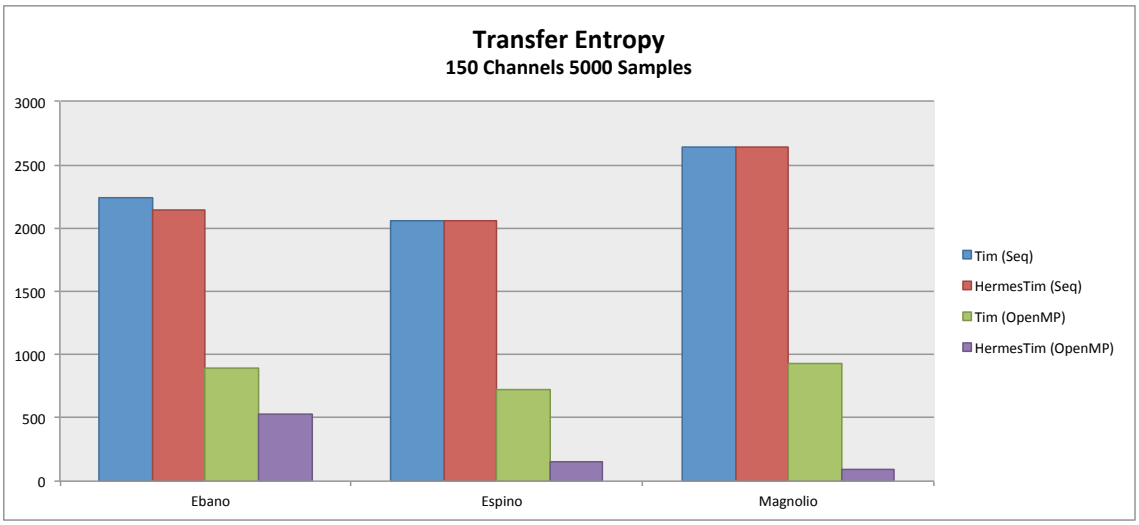

Figure 6.5: TE execution times from MATLAB: Scenario 2.

ideal speedup, when the Tim library have a lower gain.

Furthermore, The time measured using the sequential version of HermesTim library is lower than the time measured using the sequential version of Tim library. Adding that the Speedup is compared with the associated sequential version of the same library, the improvement achieved with the new library is better in terms of absolute time.

In Espino the speedup obtained is greater than the ideal speedup, 12, in many cases. This incident is related with Hyper-Threading technology[17] that the processors of Espino implement. This technology gives a notion of having double number of processor to the operative system, but only few components of the processor are duplicated. The operative system plays with all processors, and in the case of OpenMP, a thread is created for each processor. The trick is when a thread is waiting for a resource, the logic of the processor gives the control to the other thread associated with this core, if this is ready to execute, this process is done by processor transparently. In the case of Espino, It has two processor with six cores each, but give the notion of twenty-four cores to the operative system. The use of hyper-threading could give worst or better result depending in the process or process to execute. In this work, the speedup achieved in the tests done to transfer entropy is greater than the ideal speedup, in these cases Hyper-threading gives a better performance.

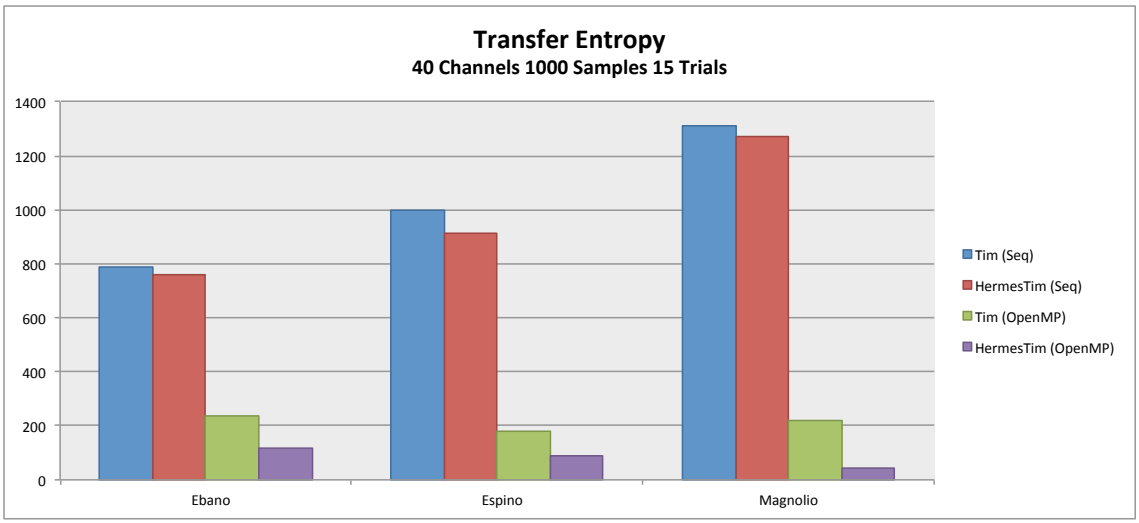

Figure 6.6: TE execution times from MATLAB: Scenario 3.

Chapter 7

Conclusions

After analysing Tim and Pastel, that were the libraries used by HERMES to compute the information theoretic indexes, these have proven to have a very complex structure and to be very optimised, because of it are not necessary to be modified.

A slight improvement is gotten writing the channel pair selection loop in C++ and not in MATLAB language. This demonstrates the difference of performance between the implementation of a loop region in C++ and the implementation of it in MATLAB. This gain is equal to the decrease of time of the sequential version of Tim compared with the sequential version of HermesTim.

Tim and Pastel are libraries with a high level of optimisation and parallelised algorithms. But the obtained performance of these libraries when are used to compute the information theoretic indexes, like mutual information or transfer entropy, are worse than it is expected, specially with the speedup achieved in many cores systems. Studying in depth the reason for these poor results it was concluded that may be caused by the small size of the work unit used into the parallels regions. So it was decided to use a larger work unit in the parallels regions.

In order to use a larger work unit in parallel regions, a new library that receives the name of HermesTim has been created. This new library acts as gateway between HERMES and the Tim library, giving a easy way to use the functions from HERMES. This library avoids loops implemented in MATLAB and it adapts the Tim functions to the HERMES needs. This results in an library that is easy to use from MATLAB and improves the performance to compute the information theoretic indexes in multi-core systems.

The parallelisation of the algorithm using a large work unit gives improvements of speedup. The speedup obtained is around 70% of ideal speedup of each machine in the scenarios that uses data sets without trials. In a first moment, In scenarios with trials this new parallelisation that it is implemented in HermesTim gave lightly worst results than the Tim library. But after done an analysis to find solutions related with the distribution of work (section 5.4), were found new solutions that were added to the HermesTim library. These solutions includes the use of *collapse* clause and nested parallel regions. These solutions improved the speedup in a great way. The TE tests that contain trials reaches $\sim 85\%$ (10,35) of maximum speedup

in Espino, $\sim$ 63% (30,24) in Magnolio and $\sim$ 79% (6,38) in Ebano using the new library HermesTim, and using the original Tim library the speedup was of 5,50 in Espino, 6,05 in Magnolio and 3,33 in Ebano for the same scenarios.

In conclusion, the HermesTim library adapts the Tim library to the HERMES needs, taking advantage that HERMES always computes the information theoretic indexes for a set of time series. HermesTim adds a new level of parallelisation where each signal pair is the work unit. This new library gives a big performance improvement in terms of time and scalability in multi-core systems. With the HermesTim library, the computation of the mutual information indexes is between 2 and 8 times faster, and the computation of transfer entropy is between 2 and 12 times faster, getting the lower profit with the scenarios with the smaller data set sizes.

7.1 Future Work

As future work:

- Analyse in depth because the speedup is so diverse for different CPU architectures.
- Add more information theoretic indexes offered by the Tim library, that are used by HERMES, in HermesTim, minimising the necessary time to compute them. (Partial mutual information, partial transfer entropy, etc).
- Propose other implementations of the information theoretic indexes that can be vectorised and can take advantage of the use of GPU acceleration.

Appendix A

Compile and Install

Requirements to install HermesTim are:

- Boost $\geq$ 1.45
- Pastel 1.2.0
- Tim 1.2.0
- Cmake
- premake 4

HermesTim offers a CMake based compilation system. This make easier the compilation in different operative systems: Windows, OS X y Linux. But to use the compilation system is neccesary to have compiled the Tim and Pastel libraries. The two first sections explain how compile the Pastel and Tim libraries. It is neccesary to apply the path included with HermesTim to avoid linking problems when Tim library is being compiled.

A.1 Pastel Compilation

Pastel is a geometry and computered graphics library implemented in C++. It is forms of various sub-libraries, but for get HermesTim is not neccesary to compile all. If you are using the path included with HermesTim distribution then it is not neccesary to modify the libraries to compile.

A.1.1 Configuration

The first step is to configure the Pastel library. Tim and Pastel use Premake configuration system, this allows to create multiplatform compilation projects or Unix Makefiles.

To configure it the configuration file must be modified, this file is inside the pastel folder. The file is named `premake4.lua`. You can open the file with any plain

text editor. Within the file is neccesary to change the route to the libraries and/or headers needed to build Pastel. The path for each is identifier by one line in the file, and this are:

```
boostIncludeDir = "../boost_1_45_0"

-- The directory of the SDL library's header files.
-- The includes are of the form 'SDL.h'.
sdlIncludeDir = ""
sdlLibraryDir = ""

-- The directory of the GLEW library's header files.
-- The includes are of the form 'glew.h'.
glewIncludeDir = ""
glewLibraryDir = ""
```

It is not neccesary to fill the Matlab include definitions, because the MATLAB interface of Pastel library are not used with HermesTim.

A.1.2 Compilation

The Premake compilation system used by Pastel allows the creation of *Unix Makefile* or *Visual Studio Projects*. The first is used in Unix based systems like OS X and Linux, although it can be used in Windows systems; the second is only used in Windows systems.

Unix Makefiles

Unix Makefiles gives an easy way to compile files or projects, normally it is included in Unix systems, but it can be used from Windows.

To generate Unix Makefiles from Pastel library, first you must be in Pastel source folder, then execute this command

```
$ premake4 gmake
```

The generated Makefiles are in *build/gmake* folder within Pastel folder.

Next step is to compile the library. Pastel comes with four different build configurations: *debug*, *develop*, *release* and *release-without-openmp*. It is recommended to use *release* configuration to build HermesTim library. The last step is built the library, this is done executing the next command from a terminal within *build/gmake* folder:

```
$ make config=release
```

Visual Studio Projects

To generate Visual Studio execute:

```
premake4 visual
```

Then the project can be opened from Visual Studio IDE.

A.2 Tim Compilation

Only the TimCore library is needed to be able to build HermesTim. If you are using the path included with HermesTim then it is not neccesary to modify the libraries to compile.

A.2.1 Configuration

The first step is to configure the Tim library, this procces is similar to the process followed to configure and compile Pastel library. Tim and Pastel use Premake configuration system, this allows to create multiplatform compilation projects or Unix Makefiles.

To configure it the configuration file must be modified, this file is inside the Tim folder, and it is named `premake4.lua`. You can open the file with any plain text editor. Within the file it is neccesary to set the routes to the libraries and/or headers specified inside the configuration file to build Tim. The path for each requirement is identified by one line in the file, and these are:

```
-- The directory of the Pastel library's source code.
-- The includes are of the form 'pastel/sys/array.h'
pastelIncludeDir = "../pastel-1.2.0"
pastelLibraryDir = "../pastel-1.2.0/build/gmake/lib"

-- The directory of the Boost library's source code.
-- The includes are of the form 'boost/static_assert.hpp'.
boostIncludeDir = "../boost_1_45_0"

-- The directory of the SDL library's header files.
-- The includes are of the form 'SDL.h'.
sdlIncludeDir = ""
sdlLibraryDir = ""
```

It is not neccesary to fill the Matlab include definitions, because Tim MATLAB interface libraries are not used with HermesTim.

A.2.2 Compilation

The Premake compilation system used by Tim allows the creation of *Unix Makefile* or *Visual Studio Projects*. The first are used in Unix based systems as OS X and Linux, although it can be used in Windows systems; the second are only used in Windows systems.

Unix Makefiles

Unix Makefiles gives an easy way to compile files or projects, normally it is included in Unix systems, but it can be used from Windows.

To generate Unix Makefiles from Tim library, first you must be in Tim source folder, then execute this command from a terminal:

```
$ premake4 gmake
```

The generated Makefiles are in *build/gmake* folder within Tim folder.

Next step is to compile the library. Tim comes with four different build configurations: *debug*, *develop*, *release* and *release-without-openmp*. It is recommended to use *release* configuration to build HermesTim library. The last step is built the library, this is done executing the next command from a terminal within *build/gmake* folder:

```
$ make config=release
```

Visual Studio Projects

To generate Visual Studio projects execute:

```
premake4 visual
```

Then the project can be opened from Visual Studio IDE.

A.3 HermesTim Compilation

HermesTim uses CMake as pre-compilation system. With CMake is possible to create *Unix Makefiles*, *Eclipse Projects*, *Visual Studio Projects*, etc, to compile the library.

Although it is possible to use CMake from console, this manual only explain the method using the CMake-gui tool, that it is a GUI for CMake system. Normally Cmake-gui is included with CMake distribution, but can be download from `http://www.cmake.org`.

Is recommended to create a folder outside HermesTim source folder to build the library.

A.3.1 Build process

1. Run Cmake-gui program, and configure source code location and build location.

2. Press configure button.

3. In the generator pop-up, that will appear, select the generator to use: Unix Makefile, Eclipse project, XCode, Visual Studio Project; and select the compilers to use. Then accept the changes.

4. When the project is configured you must fill the following options:

   - CMAKE_BUILD_TYPE: The configuration type. It is possible to select between OPENMPDEBUG, FULLOPENMPDEBUG, DEBUG for debug configuration; RELEASE, OPENMPRELEASE, FULLOPENMPRELEASE for release configurations; and RELWITHDEBINFO, OPENMPRELWITHDEB, FULLOPENMPRELWITHDEB for optimized version with debug information. The different configurations without the prefix are sequential, with the prefix OPENMP- are parallelised in HermesTim side but not in Tim and Pastel libraries, and with FULLOPENMP- prefix is the parallelized configuration for HermesTim, Tim and Pastel libraries. It is neccesary to set the Tim and Pastel libraries location according to this.
   - BOOST_1_45_DIR: Path variable of BOOST includes.
   - PASTEL_DIRECTORY: Path variable to Pastel source folder.
   - PASTEL_LIB_DIR: Path variable to Pastel build library. It is neccesary that point to the correct version of Pastel library: parallelised or sequential, according to the build type variable.
   - TIM_DIRECTORY: Path variable to Tim source folder.
   - TIM_LIB_DIR: Path variable to Tim build library. It is neccesary that point to correct version of Tim: parallelised or sequential according to the build type variable.
   - MATLAB: If selected the MATLAB interface will be built. It is neccesary to re-run configure when select it, because it is neccesary to define the MATLAB libraries locations, defined by:
     - MATLAB_INC_DIR: Path variable to matlab external includes folder. $MATLAB/external/include.
     - MATLAB_LD_DIR: Path variable to MATLAB libraries. $MATLAB/bin/$ARCH/.
     - MATLAB_LD_SYS_DIR: Path variable to MATLAB sys libraries. $MATLAB/sys/os/$ARCH/.

5. Re-run configure. If any problem is presented then press Generate button.

6. Use the tools associated with the generator selected to build the library.

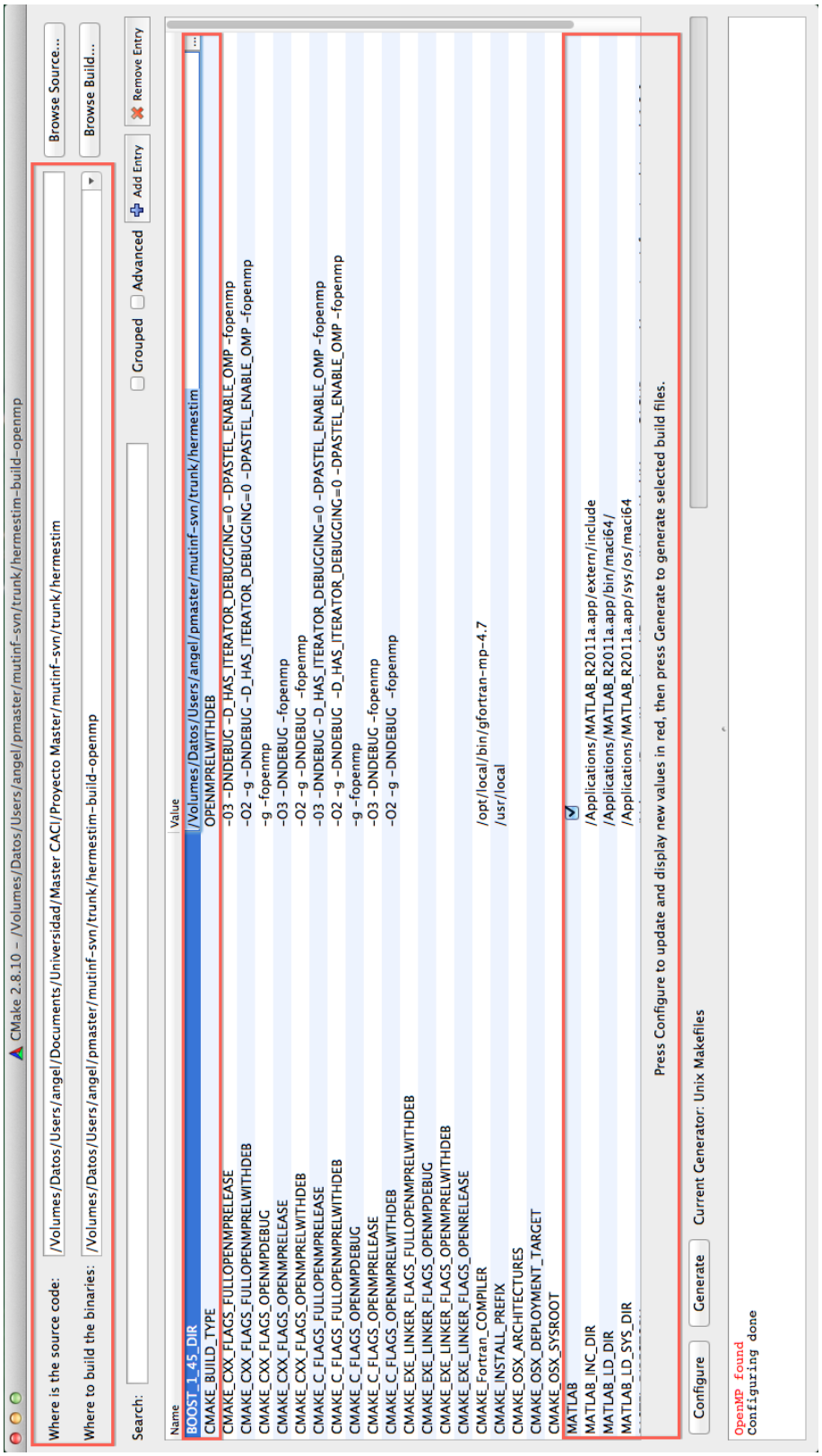

Figure A.1: HermesTim CMake-Gui configuration.

Acronyms

API application programming interface. 29, 34, 47

EC effective connectivity. 1, 3

EEG electroencephalography. 3

FC functional connectivity. 1, 3

GS generalized synchronization. 3

GUI graphic user interface. 3, 15

HERMES measure synchronisation tools, (from spanish, HERramientas de MEdidas de Sincronización). 3, 16, 19, 22, 27, 49

IDE integrated development environment. 15

KDE Kernel Density Information. 9, 10

MATLAB MATrix LABoratory. 15, 16

MEG magnetoencephalography. 1–3, 19

MI Mutual Information. 2, 7, 9, 23, 49

PS phase synchronization. 3

TE transfer entropy. 2, 10, 49, 59

Bibliography

[1] 2011. URL: http://www.cs.tut.fi/~timhome/tim/tim.htm.

[2] Bernard. *Density Estimation for Statistics and Data Analysis (Chapman & Hall/CRC Monographs on Statistics & Applied Probability)*. 1st ed. Chapman and Hall/CRC, Apr. 1986. ISBN: 0412246201. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0412246201.

[3] S. L. Bressler. "Large-scale cortical networks and cognition". In: *Brain Research Reviews* 20 (1995), pp. 288–304.

[4] G. Buzsáki. *Rhythms of the Brain*. Oxford University Press, USA, 2006. ISBN: 9780195301069. URL: http://books.google.es/books?id=Pkw7ltcn6ooC.

[5] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007. ISBN: 9780262533027.

[6] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. New York, NY, USA: Wiley-Interscience, 1991. ISBN: 0-471-06259-6.

[7] Ringo Doe. *Pastel Webpage*. 2011. URL: http://kaba.hilvi.org/pastel-1.2.0/pastel.htm.

[8] Free S. Foundation. *GNU GCC Manual*. 2008. URL: http://gcc.gnu.org/onlinedocs/gcc-4.4.7/gcc/.

[9] Peter Grassberger. "Finite sample corrections to entropy and dimension estimates". In: *Physics Letters A* 128.6–7 (1988), pp. 369 –373. ISSN: 0375-9601. DOI: http://dx.doi.org/10.1016/0375-9601(88)90193-4. URL: http://www.sciencedirect.com/science/article/pii/0375960188901934.

[10] H. Herzel, A.O. Schmitt, and W. Ebeling. "Finite sample effects in sequence analysis". In: *Chaos, Solitons Fractals* 4.1 (1994). ¡ce:title¿Chaos and Order in Symbolic Sequences¡/ce:title¿, pp. 97 –113. ISSN: 0960-0779. DOI: http://dx.doi.org/10.1016/0960-0779(94)90020-5. URL: http://www.sciencedirect.com/science/article/pii/0960077994900205.

[11] Hanspeter Herzel and Ivo Große. "Measuring correlations in symbol sequences". In: *Physica A: Statistical Mechanics and its Applications* 216.4 (1995), pp. 518 –542. ISSN: 0378-4371. DOI: http://dx.doi.org/10.1016/0378-4371(95)00104-F. URL: http://www.sciencedirect.com/science/article/pii/037843719500104F.

[12] Shunsuke Ihara. *Information theory for continuous systems*. Singapore: World Scientific, 1993.

[13] A. Kolmogorov. "Logical basis for information theory and probability theory". In: *Information Theory, IEEE Transactions on* 14.5 (1968), pp. 662–664. ISSN: 0018-9448. DOI: 10.1109/TIT.1968.1054210.

[14] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. "Estimating mutual information." In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 69.6 Pt 2 (June 2004). ISSN: 1539-3755. URL: http://view.ncbi.nlm.nih.gov/pubmed/15244698.

[15] N. N. Leonenko L. F. Kozachenko. "Sample Estimate of the Entropy of a Random Vector". In: *Problems Of Information Transmission* 23.2 (1987), pp. 95–101.

[16] Henry Markram. "The Blue Brain Project". In: *Nature Reviews Neuroscience* 7.2 (Feb. 2006), pp. 153–160. ISSN: 1471-003X. DOI: 10.1038/nrn1848. URL: http://dx.doi.org/10.1038/nrn1848.

[17] Deborah T. Marr et al. "Hyper-Threading Technology Architecture and Microarchitecture". In: *Intel Technology Journal* 6.1 (Feb. 2002), pp. 4–15. ISSN: 00419907.

[18] Young-Il Moon, Balaji Rajagopalan, and Upmanu Lall. "Estimation of mutual information using kernel density estimators". In: *Phys. Rev. E* 52.3 (Sept. 1995). DOI: 10.1103/PhysRevE.52.2318. URL: http://dx.doi.org/10.1103/PhysRevE.52.2318.

[19] Guiomar Niso et al. "HERMES: Towards an Integrated Toolbox to Characterize Functional and Effective Brain Connectivity". English. In: *Neuroinformatics* (2013), pp. 1–30. ISSN: 1539-2791. DOI: 10.1007/s12021-013-9186-1. URL: http://dx.doi.org/10.1007/s12021-013-9186-1.

[20] OpenMP Architecture Review Board. *OpenMP Application Program Interface*. Specification. OpenMP Architecture Review Board, 2011. URL: http://www.openmp.org/mp-documents/OpenMP3.1.pdf.

[21] Thomas Schreiber. "Measuring Information Transfer". In: *Phys. Rev. Lett.* 85 (2 2000), pp. 461–464. DOI: 10.1103/PhysRevLett.85.461. URL: http://link.aps.org/doi/10.1103/PhysRevLett.85.461.

[22] Claude E. Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656. URL: http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.

[23] F. Varela et al. "The brainweb: phase synchronization and large-scale integration." In: *Nature reviews. Neuroscience* 2.4 (Apr. 2001), pp. 229–239. ISSN: 1471-003X. DOI: 10.1038/35067550. URL: http://dx.doi.org/10.1038/35067550.

[24] Raul Vicente et al. "Transfer entropy—a model-free measure of effective connectivity for the neurosciences". English. In: *Journal of Computational Neuroscience* 30.1 (2011), pp. 45–67. ISSN: 0929-5313. DOI: 10.1007/s10827-010-0262-3. URL: http://dx.doi.org/10.1007/s10827-010-0262-3.